L Number	Hits	Search Text	DB	Time stamp
-	555	raid and ((receiv$4 and transfer$4 and execut$4) with (request or command))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/15 15:41
-	462	(raid and ((receiv$4 and transfer$4 and execut$4) with (request or command))) and host	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/27 10:45
-	456	((raid and ((receiv$4 and transfer$4 and execut$4) with (request or command))) and host) and storage	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/29 17:09
-	458	((raid and ((receiv$4 and transfer$4 and execut$4) with (request or command))) and host) and storage	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/29 17:09
-	871	((((receiv$4 same transfer$4 same execut$4) same (request or command))) same host) and storage	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/30 08:58
-	8	5548788.URPN.	USPAT	2003/10/30 08:59
-	27	("4371929" | "4394732" | "4755928" | "4843544" | "4870643" | "4922418" | "4965801" | "4975829" | "5101490" | "5133060" | "5148432" | "5163132" | "5175822" | "5175825" | "5179704" | "5191581" | "5206943" | "5233692" | "5237660" | "5241630" | "5247622" | "5276806" | "5371855" | "5375227" | "5390186" | "5418925" | "5459856").PN.	USPAT	2003/10/30 09:01
-	6975	raid	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/30 09:09
-	94	(((((receiv$4 same transfer$4 same execut$4) same (request or command))) same host) and storage) and raid	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/03/24 17:20
-	20	((raid and ((receiv$4 same transfer$4 same execut$4) with (request or command))) same host) and storage	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/30 09:10
-	74	((((((receiv$4 same transfer$4 same execut$4) same (request or command))) same host) and storage) and raid) not (((raid and ((receiv$4 same transfer$4 same execut$4) with (request or command))) same host) and storage)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/03 13:06
-	422	honda-k.in.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/30 09:26

-	155	honda-kiyoshi.in.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/30 09:26
-	2	(((((receiv$4 same transfer$4 same execut$4) same (request or command))) same host) and storage) and honda-kiyoshi.in.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/30 09:31
-	56	(((((receiv$4 same transfer$4 same execut$4) same (request or command))) same host) and storage) and hitachi.as.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/30 10:28
-	1514	redundant adj array adj inexpensive	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/30 10:00
-	1	(redundant adj array adj inexpensive) WITH host with (receiv$4 near3 request)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/10/30 10:08
-	36	(((((receiv$4 same transfer$4 same execut$4) same (request or command))) same host same first) and storage) and raid	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/03 15:00
-	6	((plurality or multiple) adj3 (storage)) with host with receiv$4 with trans$4 with execut$4	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/03 15:22
-	30	((plurality or multiple) adj3 (storage)) with host with ((receiv$4 or trans$4 or execut$4) near5 request)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/03 15:33
-	10	(plurality or multiple) with host with receiv$4 with trans$4 with execut$4 with request	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/03 16:46
-	2011	(disk near array).ab.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/04 14:38
-	356	((disk near array).ab.) and host and redundant	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/04 14:38
-	201	(disk near array same redundant).ab.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/04 14:42

-	40	(disk near array same redundant same transfer$4).ab.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/04 15:32
-	18	(disk near array same redundant same transfer$4).ab.	USPAT; US-PGPUB; IBM_TDB	2003/11/05 09:16
-	98	RAID1	USPAT; US-PGPUB; IBM_TDB	2003/11/05 11:27
-	2	6282610.pn.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/05 12:20
-	15	(storage adj device) near5 network near5 serial	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/05 12:22
-	350	(storage adj device) adj network	USPAT; US-PGPUB; IBM_TDB	2003/11/05 12:23
-	40	((storage adj device) adj network) and raid	USPAT; US-PGPUB; IBM_TDB	2003/11/05 12:49
-	2705	storage adj subsystem	USPAT; US-PGPUB; IBM_TDB	2003/11/05 12:53
-	1	6484229.pn.	USPAT; US-PGPUB; IBM_TDB	2003/11/05 12:54
-	1	5651132.pn.	USPAT; US-PGPUB; IBM_TDB	2003/11/05 13:05
-	6	("4761785" | "5353424" | "5373128" | "5410667" | "5418921" | "5469548").PN.	USPAT	2003/11/05 12:57
-	4348	(ring or loop or toreus) near network	USPAT; US-PGPUB; IBM_TDB	2003/11/05 13:05
-	3688	(ring or loop or toreus) adj network	USPAT; US-PGPUB; IBM_TDB	2003/11/05 13:05
-	2945	ring adj network	USPAT; US-PGPUB; IBM_TDB	2003/11/05 13:06
-	23	(storage adj subsystem) and (ring adj network)	USPAT; US-PGPUB; IBM_TDB	2003/11/05 13:20
-	21835	raid or ((plurality or many or multiple) adj (storage or disk))	USPAT; US-PGPUB; IBM_TDB	2003/11/05 13:21
-	78	(ring adj network) and (raid or ((plurality or many or multiple) adj (storage or disk)))	USPAT; US-PGPUB; IBM_TDB	2003/11/05 13:21
-	66	((ring adj network) and (raid or ((plurality or many or multiple) adj (storage or disk)))) NOT ((storage adj subsystem) and (ring adj network))	USPAT; US-PGPUB; IBM_TDB	2003/11/05 13:56
-	30	"store and forward"	USPAT; US-PGPUB; IBM_TDB	2003/11/05 13:58

-	4	(serial adj raid).ab.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/05 16:19
-	2	"20020178328"	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/06 09:39
-	5	(serial near5 raid).ab.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/05 16:35
-	1	((serial near5 raid).ab.) not ((serial adj raid).ab.)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/05 16:36
-	42	serial near5 raid	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/05 16:35
-	2152	ssa	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/05 16:37
-	85	ssa same raid	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/05 16:37
-	0	1260904.URPN.	USPAT	2003/11/06 09:14
-	1	"20020178328" and ((cooperation adj control adj information) near20 (first adj identification adj information))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/06 12:40
-	4126	daisy near chain	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/06 12:40
-	4125	daisy adj chain	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/06 12:40
-	19	(daisy adj chain) near20 raid	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/06 12:41
-	2	5768623.pn.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2003/11/06 17:39

-	2	"20020178328"	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/03/24 15:45
-	11440	(raid or (array near disk))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/03/24 17:21
-	2831	((raid or (array near disk))) and serial$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/03/24 17:21
-	114	(((read or write) near3 request) same parit$3) and (((raid or (array near disk))) and serial$3)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/03/24 17:29
-	47	((((read or write) near3 request) same parit$3) and (((raid or (array near disk))) and serial$3)) and 711/114.ccls.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/03/24 17:23
-	7	(((read or write) near3 request) same parit$3 same serial$3) and (((raid or (array near disk))) and serial$3)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/03/24 17:48
-	0	(((read or write) near3 request) same parit$3 same (serial$3 near connect$3)) and (((raid or (array near disk))))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/03/25 10:22
-	34	(raid or (redundant adj array near disks)) and (disk$2 near4 serially)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/15 15:43
-	3	(raid or (redundant adj array near disks)) and (serially near connected near3 disk)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/15 15:43
-	2	5835694.pn.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/19 12:47
-	657	raid and ((creat$3 or mak$3 or generat$3) near parity)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/21 09:48
-	442	raid and (((creat$3 or mak$3 or generat$3) near parity) same writ$3)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/21 15:31

-	66	raid and (((creat$3 or mak$3 or generat$3) near parity) same (writ$3 near request))	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/21 11:34
-	0	"data to generate parity"	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/21 14:16
-	75	raid and (((creat$3 or mak$3 or generat$3) near parity) same writ$3 same fail$4)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/21 15:31
-	2	raid same synchronously same request	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/21 16:04
-	80	raid and ((stor$3 or sav$3 or cop$3) near3 synchronously)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/21 16:05
-	14	raid and ((stor$3 or sav$3 or cop$3) near3 synchronously same request)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/21 17:24
-	122	raid and ((stor$3 or sav$3 or cop$3 or writ$3) near3 synchronously)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/21 17:24
-	33	raid and ((stor$3 or sav$3 or cop$3 or writ$3) near3 synchronously same request)	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/21 17:24

US005432922A

United States Patent [19]

Polyzois et al.

[11] **Patent Number: 5,432,922**

[45] **Date of Patent: Jul. 11, 1995**

[54] **DIGITAL STORAGE SYSTEM AND METHOD HAVING ALTERNATING DEFERRED UPDATING OF MIRRORED STORAGE DISKS**

[75] Inventors: **Christos A. Polyzois**, White Plains; **Daniel M. Dias**, Mahopac, both of N.Y.; **Anupam K. Bhide**, Foster City, Calif.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **110,467**

[22] Filed: **Aug. 23, 1993**

[51] **Int. Cl.**^{6} **G06F 13/10**; G06F 12/06

[52] **U.S. Cl.** **395/425**; 395/575; 395/600; 371/10.1; 371/40.1; 364/DIG. 1

[58] **Field of Search** 371/40.1, 11.3, 10.1; 395/575, 425, 600, 275; 364/DIG. 1

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,722,085	1/1988	Flora et al.	371/40.1
5,212,784	5/1993	Sparks	395/575
5,230,073	7/1993	Gausmann et al.	395/600
5,297,258	3/1994	Hale et al.	395/275

Primary Examiner—Rebecca L. Rudolph
Assistant Examiner—Sheela N. Nadig
Attorney, Agent, or Firm—Ronald L. Drumheller

[57] **ABSTRACT**

A fault-tolerant high performance mirrored disk subsystem is described which has an improved disk writing scheme that provides high throughput for random disk writes and at the same time guarantees high performance for disk reads. The subsystem also has an improved recovery mechanism which provides fast recovery in the event that one of the mirrored disks fails and during such recovery provides the same performance as during non-recovery periods.

Data blocks or pages which are to be written to disk are temporarily accumulated and sorted (or scheduled) into an order (or schedule) which can be written to disk efficiently, which in a preferred embodiment is in accordance with the physical location on disk at which each such block will be written. This also generally corresponds to an order which is encountered by a write head during a physical scan of a disk. The disks of a mirrored pair are operated out of phase with each other, so that one will be in read mode while the other is in write mode. Updated blocks are written out to the disk that is in write mode in sorted order, while guaranteed read performance is provided by the other disk that is in read mode. When a batch of updates has been applied to one disk of a mirrored pair, the mirrored pair switch their modes, and the other disk is updated. Preferably the updates are kept in a non-volatile memory, which furthermore advantageously may be made fault-tolerant as well.

During recovery a pair of spare alternating mirrored disks is introduced to which new updates are directed, while a background scan process copies data from the surviving disk to the new mirrored pair.

18 Claims, 8 Drawing Sheets

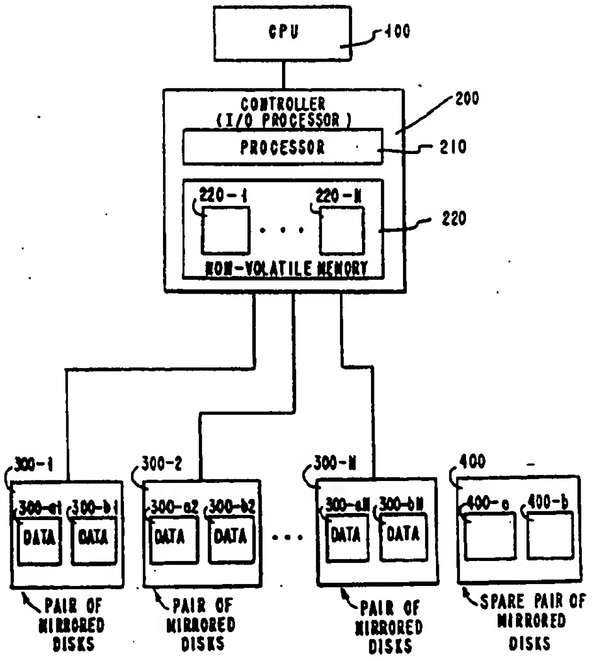


FIG. 1

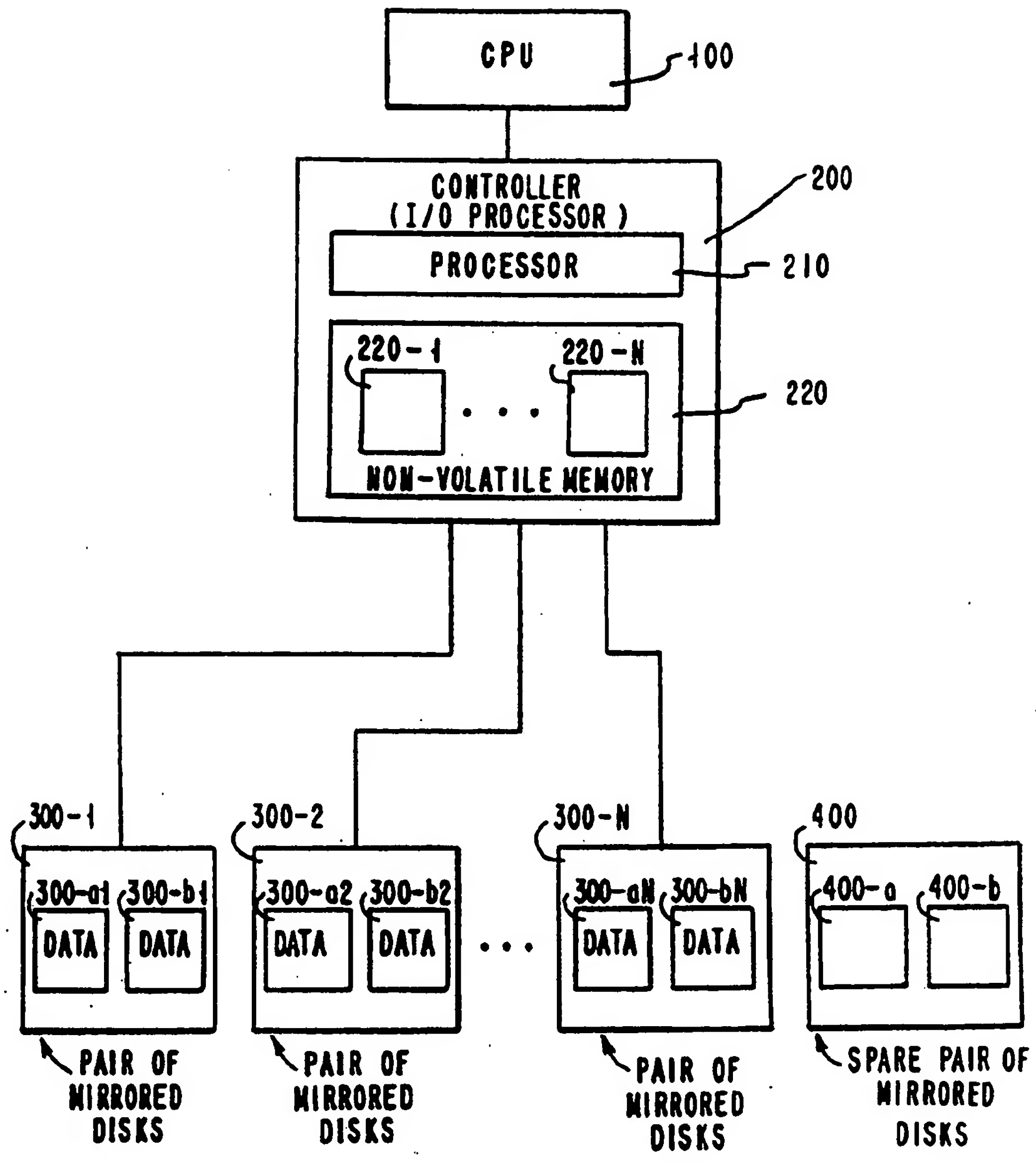

CPU
100
CONTROLLER (I/O PROCESSOR)
200
PROCESSOR
210
220-1
220-N
220
NON-VOLATILE MEMORY
300-1
300-a1
300-b1
DATA
DATA
PAIR OF MIRRORED DISKS
300-2
300-a2
300-b2
DATA
DATA
PAIR OF MIRRORED DISKS
300-N
300-aN
300-bN
DATA
DATA
PAIR OF MIRRORED DISKS
400
400-a
400-b
SPARE PAIR OF MIRRORED DISKS

FIG. 2

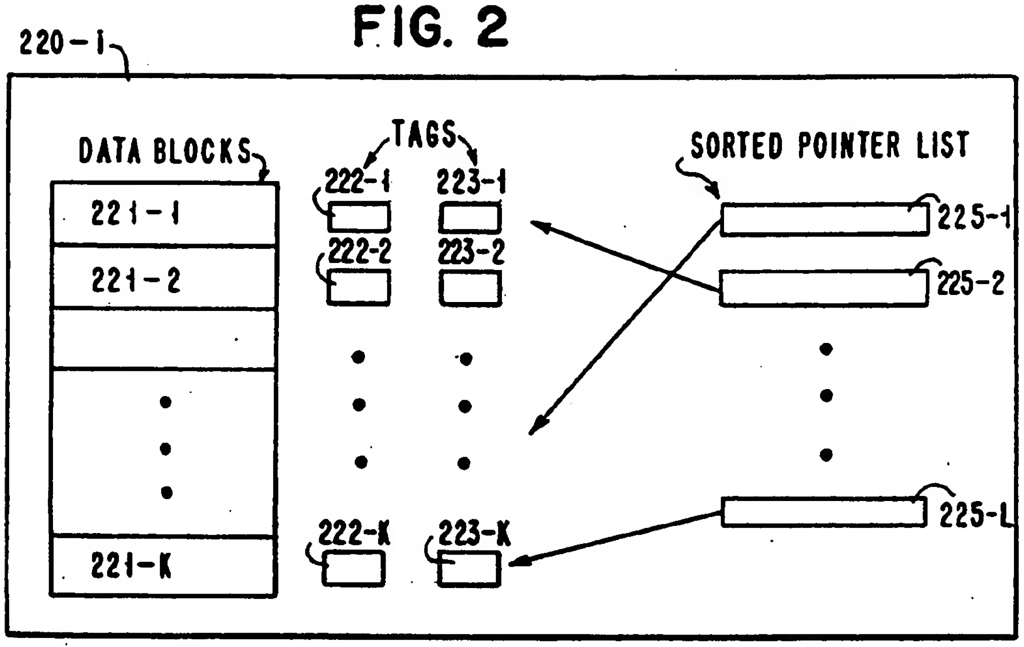

220-1
DATA BLOCKS
221-1
221-2
221-K
TAGS
222-1
223-1
222-2
223-2
222-K
223-K
SORTED POINTER LIST
225-1
225-2
225-L

FIG. 3

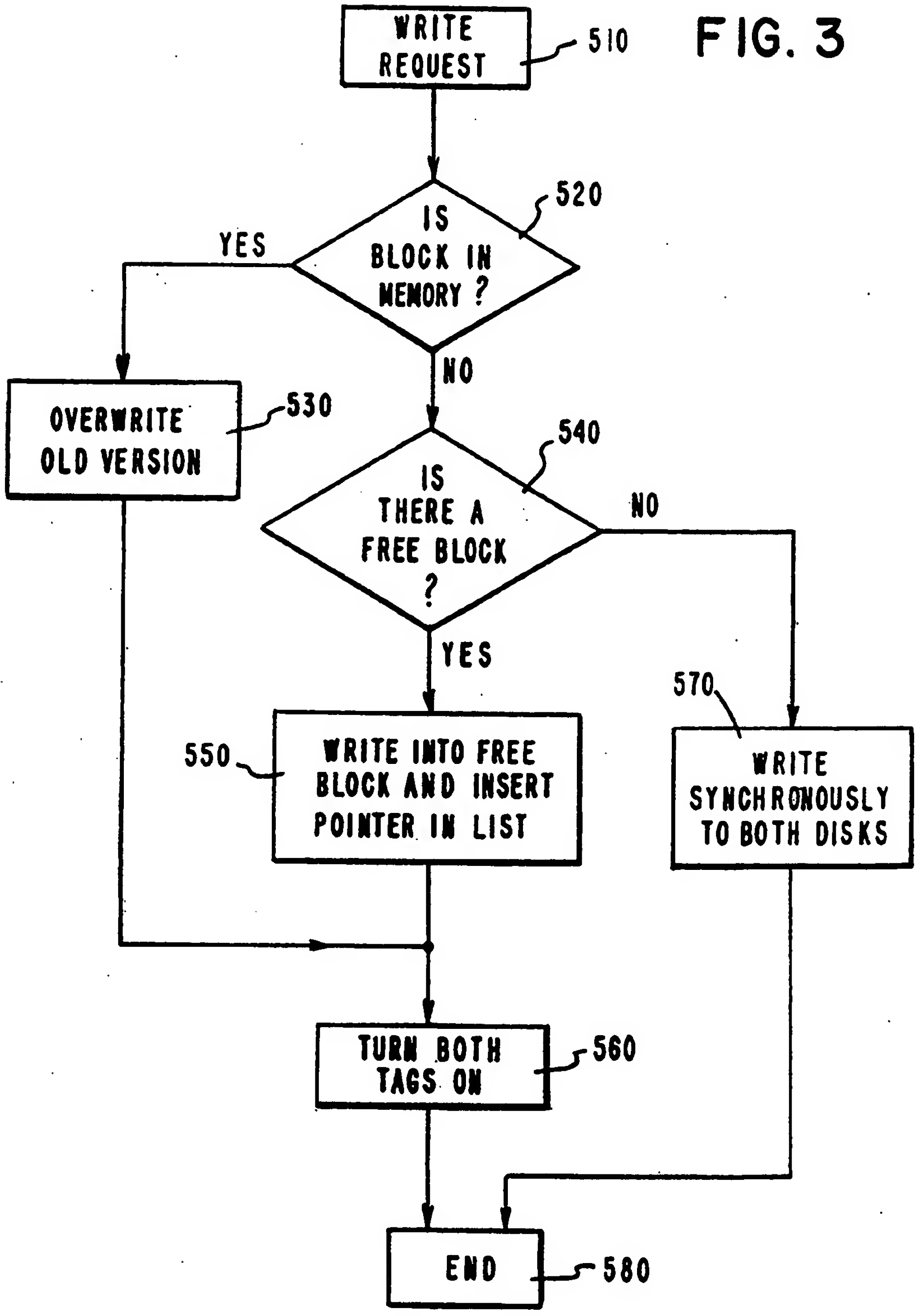

WRITE REQUEST
510
IS BLOCK IN MEMORY ?
520
YES
NO
OVERWRITE OLD VERSION
530
IS THERE A FREE BLOCK ?
540
NO
YES
WRITE INTO FREE BLOCK AND INSERT POINTER IN LIST
550
WRITE SYNCHRONOUSLY TO BOTH DISKS
570
TURN BOTH TAGS ON
560
END
580

FIG. 4

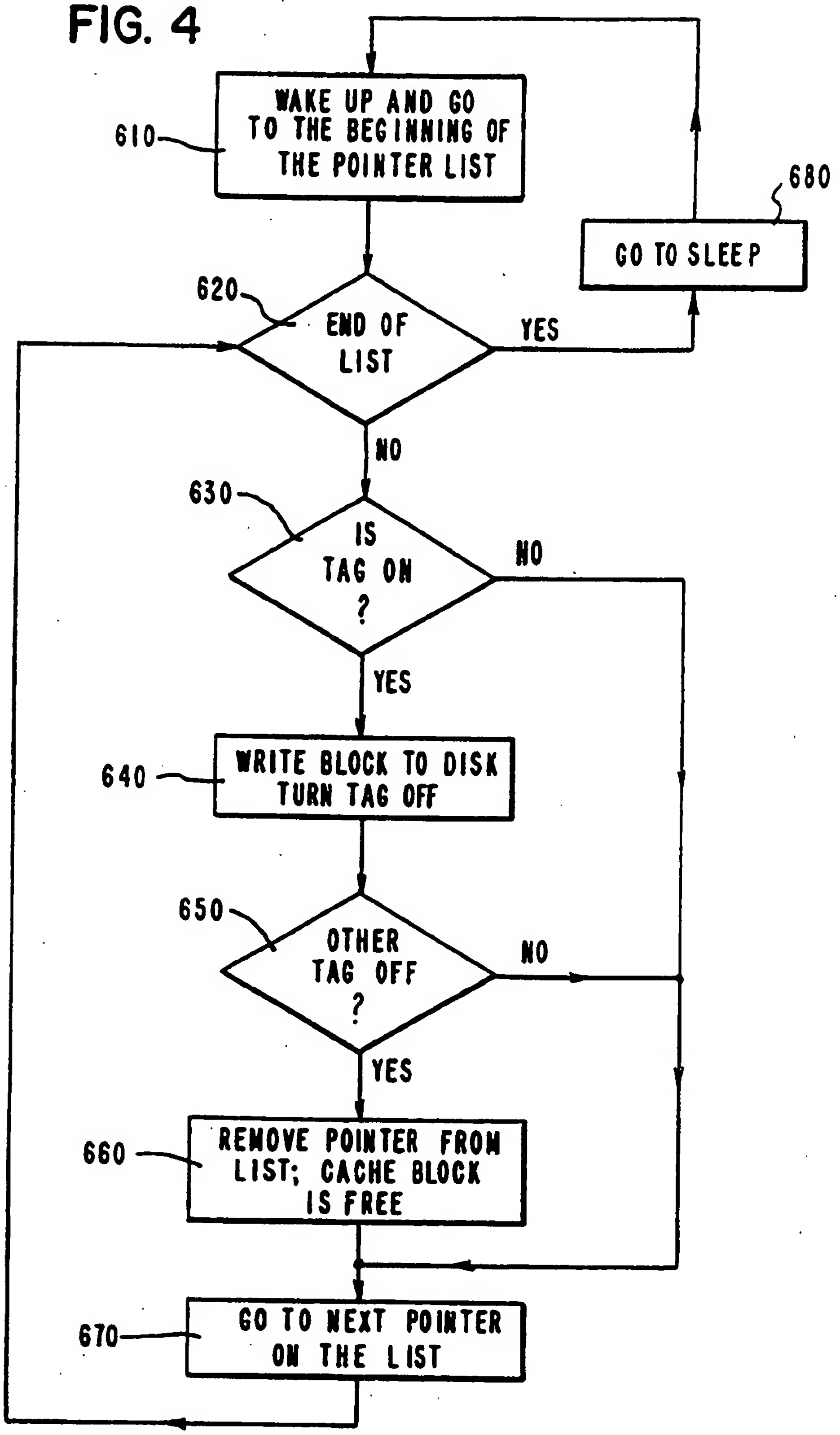

WAKE UP AND GO TO THE BEGINNING OF THE POINTER LIST
610
END OF LIST
620
YES
GO TO SLEEP
680
NO
IS TAG ON ?
630
NO
YES
WRITE BLOCK TO DISK TURN TAG OFF
640
OTHER TAG OFF ?
650
NO
YES
REMOVE POINTER FROM LIST; CACHE BLOCK IS FREE
660
GO TO NEXT POINTER ON THE LIST
670

FIG. 5

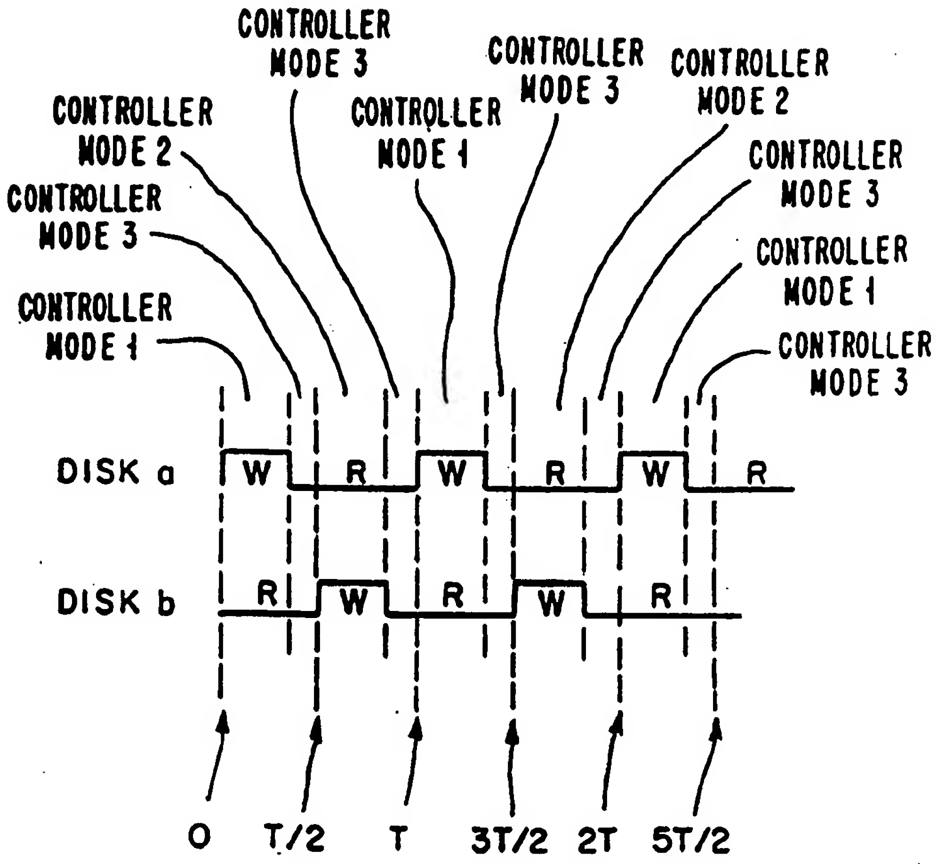

CONTROLLER MODE 1
CONTROLLER MODE 3
CONTROLLER MODE 2
CONTROLLER MODE 3
CONTROLLER MODE 1
CONTROLLER MODE 3
CONTROLLER MODE 2
CONTROLLER MODE 3
CONTROLLER MODE 1
CONTROLLER MODE 3
DISK a
DISK b
W
R
O
T/2
T
3T/2
2T
5T/2

FIG. 7

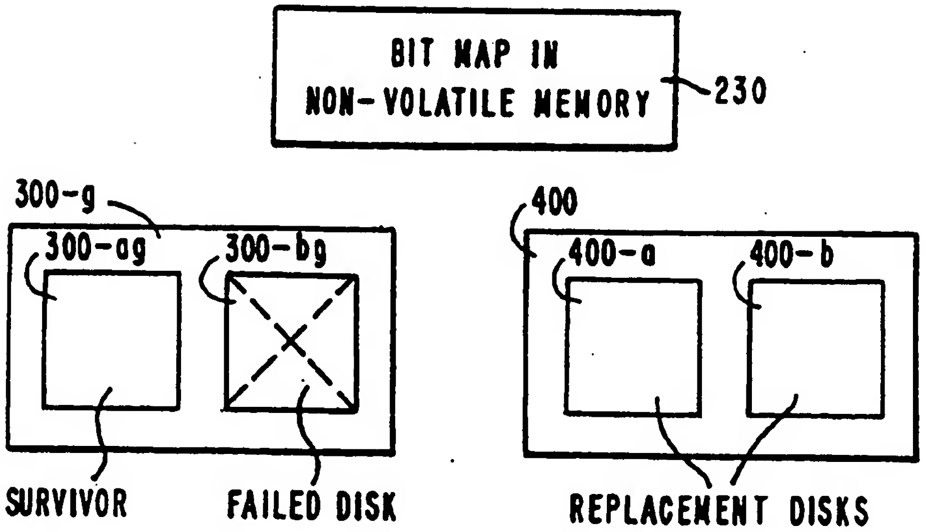

BIT MAP IN NON-VOLATILE MEMORY
230
300-g
300-ag
300-bg
400
400-a
400-b
SURVIVOR
FAILED DISK
REPLACEMENT DISKS

FIG. 6

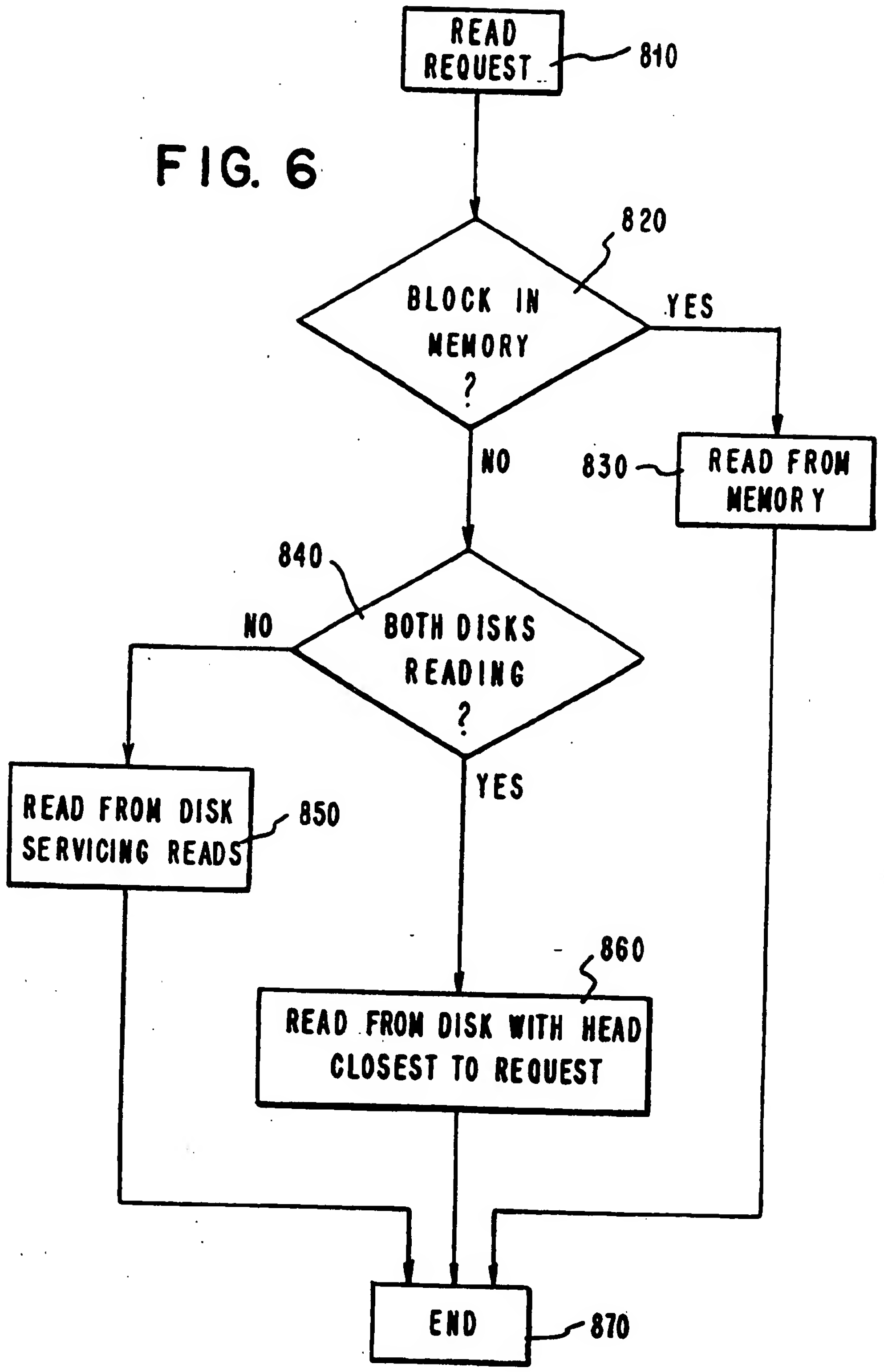
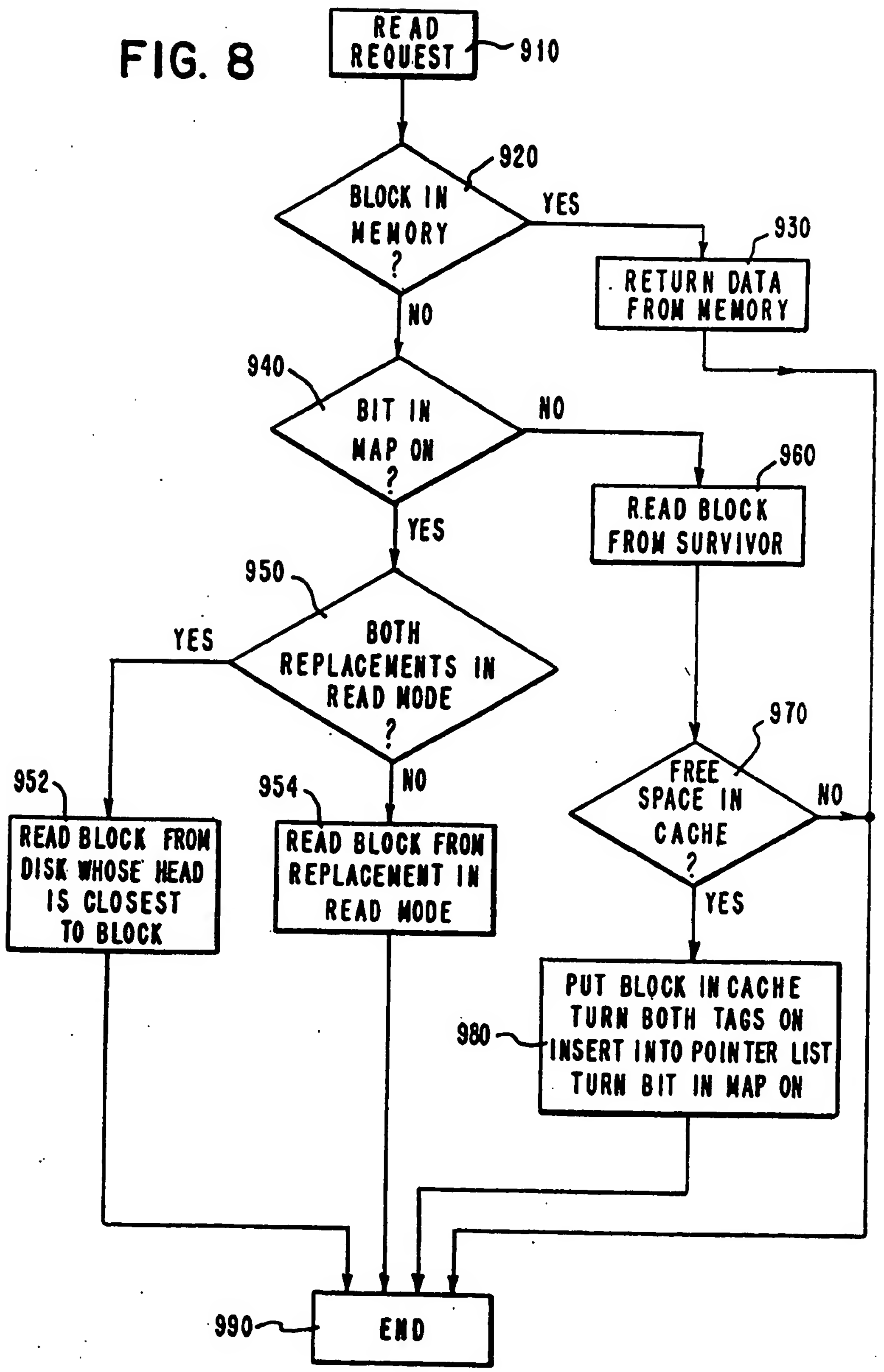

READ REQUEST
810
BLOCK IN MEMORY ?
820
YES
NO
READ FROM MEMORY
830
BOTH DISKS READING ?
840
NO
YES
READ FROM DISK SERVICING READS
850
READ FROM DISK WITH HEAD CLOSEST TO REQUEST
860
END
870

FIG. 8


READ REQUEST
910
BLOCK IN MEMORY ?
920
YES
930
RETURN DATA FROM MEMORY
NO
940
BIT IN MAP ON ?
NO
960
READ BLOCK FROM SURVIVOR
YES
950
BOTH REPLACEMENTS IN READ MODE ?
YES
NO
952
READ BLOCK FROM DISK WHOSE HEAD IS CLOSEST TO BLOCK
954
READ BLOCK FROM REPLACEMENT IN READ MODE
970
FREE SPACE IN CACHE ?
NO
YES
PUT BLOCK IN CACHE
TURN BOTH TAGS ON
INSERT INTO POINTER LIST
TURN BIT IN MAP ON
980
990
END

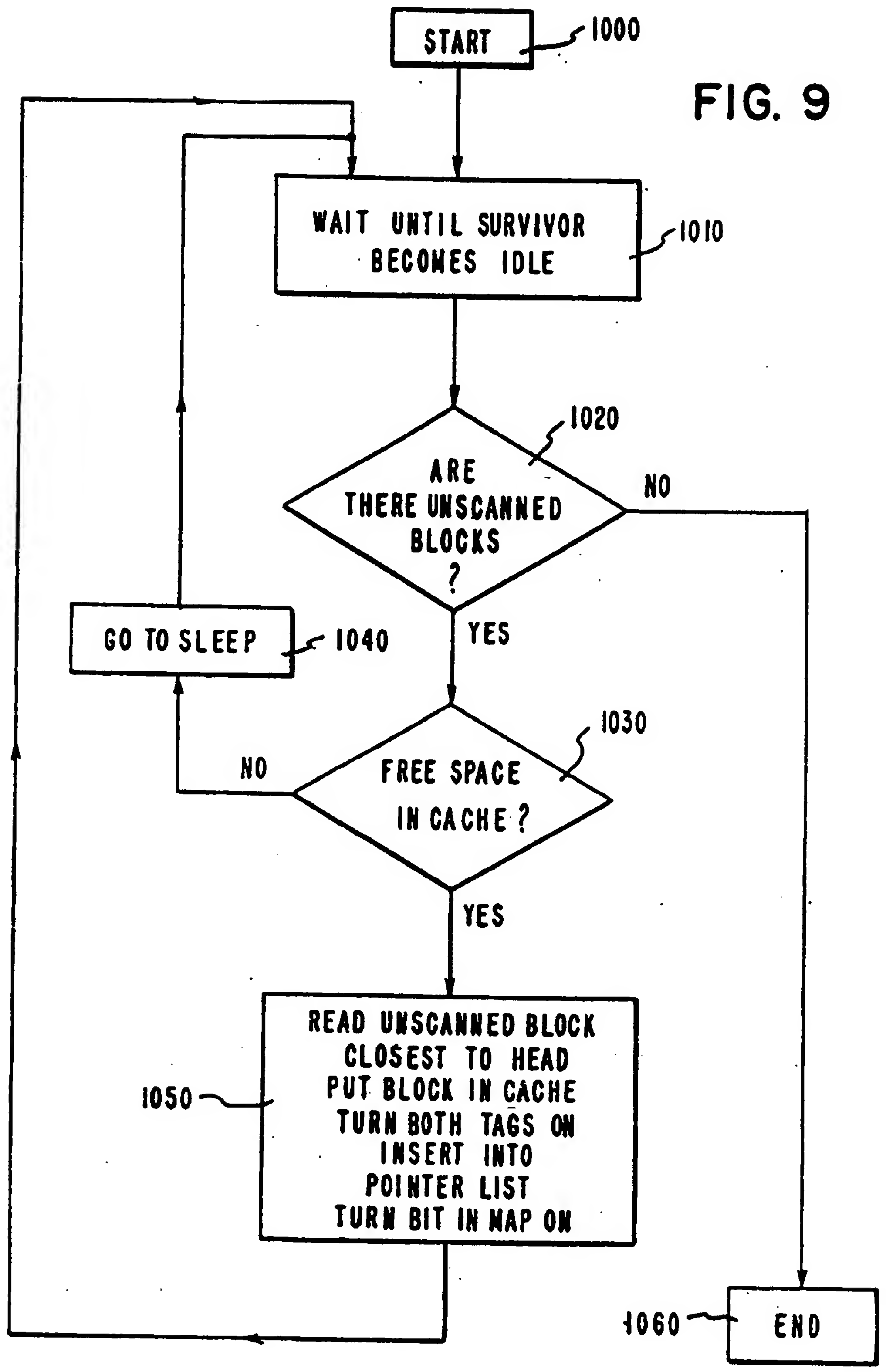

START
1000
WAIT UNTIL SURVIVOR BECOMES IDLE
1010
ARE THERE UNSCANNED BLOCKS ?
1020
NO
YES
GO TO SLEEP
1040
FREE SPACE IN CACHE ?
1030
NO
YES
READ UNSCANNED BLOCK CLOSEST TO HEAD
PUT BLOCK IN CACHE
TURN BOTH TAGS ON
INSERT INTO POINTER LIST
TURN BIT IN MAP ON
1050
1060
END

FIG. 9

DIGITAL STORAGE SYSTEM AND METHOD HAVING ALTERNATING DEFERRED UPDATING OF MIRRORED STORAGE DISKS

FIELD OF THE INVENTION

This invention relates generally to fault tolerant digital storage disk systems and more particularly to digital storage disk systems of the mirrored disk type in which reliability is provided by storing digital information in duplicate on two separate storage disks.

BACKGROUND OF THE INVENTION

As the requirements for On-Line Database Transaction Processing (OLTP) grow, high transaction rates on the order of thousands of transactions per second must be supported by OLTP systems. Furthermore, these applications call for high availability and fault tolerance. In applications such as OLTP, a large fraction of the requests are random accesses to data. Since a large fraction of the data resides on disks, the disk sub-systems must therefore support a high rate of random accesses, on the order of several thousands of random accesses per second. Furthermore, the disks need to be fault tolerant to meet the availability needs of OLTP.

Whenever a random access is made to a disk, in general the disk must rotate to a new orientation such that the desired data is under a disk arm and the read/write head on that disk arm must also move along the arm to a new radial position at which the desired data is under the read/write head. Unfortunately performance of this physical operation, and therefore random disk Input/Output (I/O) performance is not improving as fast as other system parameters such as CPU MIPS. Therefore, applications such as OLTP, where random access to data predominates, have become limited by this factor, which is referred to in this art as being disk arm bound. In systems which are disk arm bound, the disk cost is becoming an ever larger fraction of the system cost. Thus, there is a need for a disk sub-system which can support a larger rate of random accesses per second with a better price-performance characteristic than is provided by transitional disk systems.

Both mirrored disk systems and RAID disk systems (for Redundant Array of Independent Disks) have been used to provide fault tolerant disk systems for OLTP. In a mirrored disk system, the information on each disk is duplicated on a second (and therefore redundant) disk. In a RAID array, the information at corresponding block locations on several disks is used to create a parity block on another disk. In the event of failure, any one of the disks in a RAID array can be reconstructed from the others in the array. RAID architectures require less disks for a specified storage capacity, but mirrored disks generally perform better. In an article entitled "An evaluation of redundant arrays of disks using an Amdahl 5890," SIGMETRICS Conference on Measurement and Modeling of Computer Systems, pp. 74–85, Boulder, Colo., May 1990, P. Chen et al. showed that mirrored disks are better than RAID-5 disk arrays for workloads with predominantly random writes (i.e., average read/write times for mirrored disk architectures are lower than for RAID-5 architectures when random read/writes predominate). RAID-5 architecture is described, for example, by D. Patterson et al. in "A case for redundant arrays of inexpensive disks," ACM SIGMOD Int'l Conf. on Management of Data, pp. 109–116, Chicago, Ill. (June 1988). However, mir-

rored disks do require that each data write be written on both disks in a mirrored pair. Thus, it is generally accepted that mirrored disk storage systems impose a performance penalty in order to provide the fault tolerance.

In a pending patent application Ser. No. 8-036636 filed Mar. 24, 1993, assigned to the same assignee as this patent application, entitled "Disk Storage Method and Apparatus for Converting Random Writes To Sequential Writes While Retaining Physical Clustering on Disk", some of the inventors of the present invention disclosed a method for improving the performance of a single disk or a RAID array. This is done by building sorted runs of disk writes in memory, writing them to a log disk, merging the sorted runs from the log disk and applying them in one pass through the data disks with large batch writes. This method has the advantage of largely converting random writes into sequential writes. One problem with this approach, however, is that when random disk reads interrupt the batch writes, either the disk read requests are delayed while the batch is written, leading to a penalty in disk read response time, or the batch writes are interrupted by the read, leading to a large loss in write (and therefore overall) throughput. Either way the overall performance suffers so that the benefit of creating sorted runs is offset largely whenever random disk reads are needed frequently during batch write operations.

The traditional method for recovery in a mirrored disk system is to copy the data from the surviving disk of the mirrored pair onto a spare backup disk. This is typically done by scanning the data on the surviving disk, and applying any writes that come in during this process to both disks. One problem with this approach is that it produces a significant degradation of the disk system performance during recovery.

SUMMARY OF THE INVENTION

Accordingly, it is an object of this invention to improve the performance of mirrored disk systems by largely eliminating the penalty normally resulting from the need to duplicate each disk write onto both disks of a mirrored pair of disks.

It is also an object to provide a mirrored disk subsystem that has improved performance for random disk I/O by converting random disk write I/O to close to sequential I/O.

It is still another object to improve the mirrored disk throughput without a penalty in read response time.

It is also an object to improve performance during the recovery process from a failed disk, by providing guaranteed performance to disk reads and writes during recovery, while retaining fast recovery.

These and further objects and advantages are achieved by this invention by providing a fault-tolerant disk storage subsystem of the mirrored disk type in which updates (i.e., data blocks to be written) to disk are accumulated and scheduled into successive batch runs of updates, the scheduling being done to produce an ordering which can be written efficiently to the mirrored disks. The updates preferably, but not necessarily, are accumulated in a memory in the disk controller, and the scheduling is preferably, but not necessarily, done by the disk controller for the mirrored disks. Preferably the memory is either non-volatile or fault-tolerant.

In a preferred embodiment, the scheduling is done by sorting the updates in accordance with the home locations of the updates on the mirrored disks (i.e., in accordance with the positions on disk at which the updates will be written). This is an ordering which also corresponds to a scan of a disk.

The disks in each mirrored pair are then operated out of phase with each other, one being in read mode while the other is in write mode. A batch of writes is efficiently applied each time to the disk in write mode in accordance with the scheduled order. Because the updates are copied onto each disk of the mirrored pair in accordance with the physical order on the disk, good performance is achieved for applying the writes.

Random writes are thus converted to largely sequential writes to disk and clustering of data on the disk is preserved. The average time to apply a write of a block using this method is typically less than half the time to apply a random write of a block on the disk, thus largely eliminating the problem of having to write a block twice to a pair of mirrored disks.

During this time, read requests are either handled by reading the data from the memory or from the disk that is in read mode. Thus, guaranteed performance to read requests is also achieved. When a batch of updates has been applied to one of the disks of a mirrored pair (i.e., the one in write mode), the disks switch modes of operation. There may also be a period of time when both disks are in read mode between these two modes of operation. Also there may be times when both disks of a mirrored pair are in write mode, as for example during loads and other large copying operations.

Recovery from failure of one disk of a mirrored pair of disks is handled by introducing a pair of spare mirrored disks that are operated using the alternating mirrors scheme. During recovery, new writes are directed to the spare disk pair. Reads are either handled from the surviving disk or the alternating mirror spare disk pair. In the background, spare cycles are used to scan and copy data from the surviving disk to the spare alternating mirror pair. This method provides fast recovery with guaranteed performance to both read and write requests during recovery.

BRIEF DESCRIPTION OF THE DRAWINGS

These, and further, objects, advantages, and features of the invention will be more apparent from the following detailed description of a preferred embodiment and the appended drawings in which:

FIG. 1 is an overall block diagram of a preferred embodiment of this invention;

FIG. 2 illustrates a preferred organization of data in the non-volatile memory of the I/O processor;

FIG. 3 is a flow chart which shows the steps involved in processing a write request during normal operation;

FIG. 4 is a flow chart which shows the steps involved in the process of applying a batch of writes to a disk during normal operation;

FIG. 5 is a timing diagram which shows the timing relation between the two processes applying write batches to two mirrored disks;

FIG. 6 is a flow chart which shows the steps involved in processing a read request during normal operation;

FIG. 7 schematically illustrates the configuration during recovery of a failed disk;

FIG. 8 is a flow chart which shows the steps involved in servicing a read request during recovery;

FIG. 9 is a flow chart which shows the steps involved in the background process that scans the survivor disk.

DESCRIPTION OF A PREFERRED EMBODIMENT

FIG. 1 is a block diagram of a preferred embodiment of a computer system which incorporates a disk storage subsystem having mirrored storage disks which are alternately updated in a batch fashion with accumulated updates that have been sorted for efficient writing (henceforth sometimes called AMDU for Alternating Mirrors with Deferred Updates) in accordance with this invention. It includes a controller or I/O processor (IOP) **200**, a plurality of mirrored disk pairs **300-1** through **300-N**, at least one spare pair of disks **400** and a central processing unit (CPU) **100**. The controller **200** is connected to the CPU **100** and has a processor **210** and non-volatile memory **220**. For simplicity we assume that the non-volatile memory is partitioned into regions **220-1** through **220-N**, each corresponding to a mirrored pair.

Those skilled in the art will readily appreciate that the memory and controller need not constitute a separate physical subsystem as illustrated, but could instead be implemented with software running in the main computer system. Also the memory need not be non-volatile in order to achieve useful benefit from this invention and in many environments could be fault-tolerant as well (say through use of triple redundancy and a voting mechanism). The memory also need not be partitioned and there can be more than one spare pair of disks. The spare pair of disks is not used in normal operation, but is used in the event one of the mirrored disks fails.

Each mirrored disk pair consists of two disks labeled **300-a1** and **300-b1** for disk pair **300-1** (correspondingly **300-aN** and **300-bN** for disk pair **300-N**). The two disks in each mirrored pair contain basically identical data. However, as will be better appreciated from the following description, updates to each mirrored pair of disks are NOT made simultaneously as would be the case with conventional mirrored disks. In accordance with the invention, updates are accumulated instead in the non-volatile memory **220** and sorted into batches of updates, which are applied to the two disks of a pair not simultaneously, but rather first to one and then to the other. Furthermore, while the same updates are eventually applied to each of the two disks of a pair (except for updates that have become obsolete because they have been further updated in non-volatile memory **220** before being applied to both disks), the batches of updates made to each disk individually generally are not identical since they are applied at different times and more recent blocks of data are included in the individual batch for each disk of a pair.

FIG. 2 shows on region of non-volatile memory **220-i** in more detail. The region has a number of data blocks labeled **221-1** through **221-k**. Corresponding to each data block are two tags labeled **222-1** and **2223-1** (for block **221-1**) through **222-k** and **223-k** (for block **221-k**). The two tags for each data block correspond to the two disks in the mirrored pair and indicate whether the corresponding disk must still write the data block.

In each region of the non-volatile memory there is also a list of pointers labeled **225-1** through **225-L**. Each pointer points to a data block in some region of non-volatile memory. The order of the pointers in the list indicates the order in which the blocks should be written on the disk to achieve efficiency.

The non-volatile memory acts as a cache for data blocks to be written to disk. Those skilled in the art will readily appreciate that this cache may be managed like any other cache, such as by using a hash table (to determine which blocks are present in the cache) and a free list (to chain free blocks). The hash table and the free list are not shown in FIG. 2.

In the preferred subsystem in accordance with this invention, there are four processes occurring during normal operation. The first process services write requests and is shown in FIG. 3. When a write request arrives (block **510**), the non-volatile memory in the controller is checked for the presence of an old version of that block (block **520** in FIG. 3). If the previous version of that block is already in the non-volatile memory region for the corresponding pair of disks, the previous version in memory is overwritten (block **530**) and both tags corresponding to that block are turned on (block **560**) to indicate that both disks must install the new version of that block.

If an old version of the block to be written is not found in the non-volatile memory, the controller looks for free space in its non-volatile memory in which to temporarily store the block to be written (block **540**). If there is a free space for the block, the data block defined by the write request is written into the free space and a pointer to the new block is inserted into the pointer list (block **550**) in a position (relative to other pointers in the list) such that the list represents a schedule for efficiently writing the pointed-to data blocks to disk. The corresponding tags are turned on to indicate that both disks must perform a write of the new data block (block **560**).

If the data block to be written is not in the non-volatile memory and there is no free block space available, the new block is written synchronously to both disks (block **570**). This situation will not occur normally (other than maybe for loads and other very large copying operations) if the non-volatile memory is large enough to absorb heavy bursts of write activity, but the action to take in the event a write request is encountered and there is no free space in the non-volatile memory must be specified anyway.

The list of pointers **225** defines an order or schedule for the data blocks (in the non-volatile memory) covered by that list. This order or schedule is created preferably such that if a disk accesses the blocks in that order, the total time to access all the blocks (and write them to disk) will be minimized. As a first approximation, the ordering may be by cylinder, so that a scan (sweep) through the entire disk can apply all updates in one pass. All blocks in a particular cylinder are written before moving on to the next cylinder. More elaborate schemes could order the blocks within a cylinder to minimize the rotation latency for the cylinder. More sophisticated schemes may take into account the combination of seek time and rotational delay. Such schemes are described, for example, by M. Seltzer et al. in "Disk Scheduling Revisited", Winter 1990 USENIX, pp. 313-323, Washington, D.C. (Jan. 1990).

There are two processes (one for each disk), which periodically wake up and apply the updates pending in non-volatile memory to the corresponding disk. The logic for applying the updates is identical for the two processes and is shown in FIG. 4. When the process wakes up (block **610**), it goes to the beginning of the pointer list and traverses the pointer list examining each block in order. In each step it checks to see if it has reached the end of the list (block **620**). If so, it goes to sleep (block **680**) and wakes up for the next period. If there are more blocks, it checks the tag of the current block in the pointer list (block **630**). If the tag corresponding to the process's disk is off, the process moves to the next pointer in the list (block **670**). If the tag is on, the process writes the block on disk and turns the tag off (block **640**). After turning the tag off, it checks the other tag (block **650**). If the other tag is on, the process moves to the next pointer in the list (block **670**). If the other tag is off, both disks have applied the update, so the pointer is removed from the list (block **660**). The block is still valid (a read to that block will still get a cache hit), but it is free and can be overwritten by a subsequent write of any block. Then the process moves to the next pointer in the list (block **670**).

The two processes applying the updates to the two disks have the same logic and preferably the same period, i.e., the time that elapses between two consecutive activations of the write phase of the process preferably is the same for the two processes. This period is called T in the described embodiment. The two processes need not be synchronized, but they are illustrated at a phase difference of 180 degrees in FIG. 5. The process for disk a wakes up at times 0, T, 2T, 3T etc., and begins to write a batch of updates to a disk a until completed and then switches to read mode, while the process for disk b wakes up at times T/2, T+T/2, 2T+T/2, 3T+T/2, etc., and begins to write a batch of updates to disk b until completed and then switches to read mode. FIG. 5 illustrates this with a timing diagram, where the high value indicates time periods during which the writing process is active for the corresponding disk. While the writing process is asleep (inactive), the corresponding disk may service random read traffic.

As illustrated in FIG. 5, this results in three different controller modes, namely controller mode **1** where disk a is in write mode and disk b is in read mode, controller mode **2** where disk b is in write mode and disk a is in read mode, and controller mode **3** where both disk a and disk b are in read mode. As mentioned earlier in connection with the description of block **570** of FIG. 3, there is also a controller mode **4** where both disk a and disk b are in write mode. Controller mode **4** cannot occur so long as the batch write completes each time in less than time T/2. The system is preferably designed so that the situation where both disks are in write mode simultaneously is largely avoided, which is done by making the design such that batch writes will complete in less than time T/2.

Keeping the two processes at phase difference of 180 degrees ensures that if writes can be applied in less than half a period, there is always one disk arm dedicated to servicing random reads, which allows batches to become large (so that writes can gain efficiency), without hurting response time for reads. The period T is a system-dependent parameter, primarily determined by the amount of memory available, since the writes accumulating in a period T should fit in memory.

The logic for the process servicing read requests is shown in FIG. 6. When a read request arrives (block **810**), the non-volatile memory is checked (block **820**) for the presence of the block to be read. If the block is in memory, it is returned immediately (block **830**). If it is not in memory, a check is made (block **840**) as to whether both disks are currently servicing read requests (i.e., whether the controller is in controller mode **3**). If not, the request is served by the disk that is currently in read-only mode (block 850), i.e., the disk whose write process is inactive. If both disks are in read-only mode (i.e., controller mode 3), the request may be serviced by either disk, but preferably will be serviced by the disk whose heads are closest to the target block (block 860).

Those skilled in the art will readily appreciate that some routine synchronization (e.g., latching) is required to preserve the integrity of the shared data structures (e.g., tags, pointer list) accessed simultaneously by more than one of the above processes. Also, the pointer order that minimizes the time to write a batch may be different for the two disks, since each disk generally writes to a disk a different subset of the blocks stored in the non-volatile memory.

Operation under a failure scenario will now be described and is illustrated in FIG. 7. Assume that disk 300-bg in mirrored pair 300-g fails. Traditional recovery schemes use one replacement disk, onto which the contents of the surviving disk 300-ag are copied to replace the lost mirrored disk and therefore restore the mirrored pair. The preferred recovery scheme in accordance with this invention utilizes a pair of replacement disks 400*a* and 400*b*, rather than just a single replacement disk. When recovery completes, the disks in pair 400 are up-to-date, and the survivor 300-ag is returned to the system for other use.

For the duration of recovery, the survivor stays in read-only mode. The survivor does not get involved in servicing writes. Those skilled in the art will readily appreciate that a bit map (labeled 230 in FIG. 7) stored in the non-volatile memory can be used to keep track of which blocks remain to be retrieved from the survivor before recovery completes. The bit map has one bit per disk block and all bits are clear when recovery starts. Alternatively, the bit map can be stored in other memory components of the system.

In total, there are five processes involved in recovery. Two of the processes (one for each disk) periodically wake up and apply writes pending in the cache to the corresponding disk. The processes have the same period and are maintained at a phase difference of 180 degrees. The logic of these processes is identical to that for normal operation shown in FIG. 4. The third process services reads and is shown in FIG. 8. When a read request arrives (block 910), the memory is checked (block 920) for the presence of the block to be read. If the block is present in memory, it is returned immediately (block 930). If the block is not in the non-volatile memory, it must be read from disk. The bit map is first checked to see if the block is available on the replacement disks (block 940). If not, the read is serviced by the survivor (block 960). After the block is read from the survivor, the process checks if there is free space in the non-volatile memory (block 970). If not, the process ends. If there is free space in the non-volatile memory, the block is also placed in the non-volatile memory, a pointer is inserted in the list and both tags are turned on (block 980), so that the disks will write it in their next write phase. Furthermore, the bit in the bit map corresponding to that block is turned on to indicate that there is no longer a need to extract that block from the survivor.

If on a read request the block is not in non-volatile memory and the bit map shows that the block is available on the replacement disks, the block is read from one of the replacements. The process preferably checks show many replacements are in read mode at that moment (block 950). If only one replacement is in read

mode, the request is served by that replacement (block 954). If both replacements are in read mode, the request preferably is serviced by the disk whose arm is closest to the requested block (block 952). If the block is read from a replacement disk, there is no need to update the bit map or store the block in the cache.

The fourth process services writes during recovery and involves exactly the same steps shown in FIG. 3. In addition, in all cases, the bit corresponding to the block written is set in the bit map (if not already set) to indicate that it is no longer necessary to copy that block from the survivor.

The fifth process (shown in FIG. 9) is a background process that scans the blocks in the survivor that have not yet been written on the replacements. The process is started (block 1000) when the system enters the recovery mode and the replacement disks are activated. The process waits until the survivor becomes idle (block 1010), i.e., until there are no random read requests pending for it. Then, it checks if there are unscanned blocks (block 1020), i.e., blocks for which the bit in the bit map has not been set. If all blocks have been scanned, recovery is complete and the process terminates (block 1060); the survivor can be returned to the system for other use. If there are unscanned blocks, the process checks if there is free space in the non-volatile memory (block 1030). If not, it goes to sleep (block 1040) for a certain interval. If there is free space in the non-volatile memory, the process reads the unscanned block which is closest to the current position of the survivor's head (block 1050). The bit map is used to determine which blocks are unscanned. The block read is placed in the non-volatile memory, and both of its tags are turned on to indicate that the replacements must write the block. A pointer is also inserted in the pointer list. Furthermore, the corresponding bit in the bit map is set to indicate that the survivor does not need to scan that block again. The process then repeats the above steps (goes to block 1010). If a random read arrives, the process is suspended in block 1010 until the read completes.

Those skilled in the art will readily appreciate that there are opportunistic strategies which the survivor disk can use to further expedite the recovery process. For example, whenever the survivor disk services a random read request, it could also read any unscanned (i.e., uncopied) blocks that happen to pass under its arm while it is waiting for the disk to rotate to the targeted block. Furthermore, the process shown in FIG. 9 obviously could read more than one block at a time.

We claim:

1. A fault-tolerant disk storage subsystem for storing data blocks of digital information for a computer system, comprising:

a mirrored pair of disks for storing data blocks of digital information in duplicate on both disks of said mirrored pair; and

a controller for said mirrored pair of disks, said controller having a memory;, said controller comprising:

means for temporarily accumulating in said memory until storage thereof in duplicate on both disks of said mirrored pair a multiplicity of data blocks provided by said computer system as separate writes to the disk storage subsystem;

means for identifying each block stored in said memory that has not yet been stored on one disk of said pair and for identifying each block stored in said memory that has not yet been stored on the other disk of said pair;

means for sorting said accumulated data blocks that have not yet been stored on said one disk into an order that can be efficiently written onto said one disk in a batch run and for sorting said accumulated data blocks that have not yet been stored on said other disk into an order that can be efficiently written onto said other disk in another batch run;

means for providing a first mode of operation in which said one disk is in a write-only mode and said sorted accumulated data blocks that have not been stored on said one disk are written in batch mode onto said one disk, while said other disk serves said computer system in a read-only mode and writes from said computer system are received into said memory;

means for providing a second mode of operation in which said other disk is in a write-only mode and said sorted accumulated data blocks that have not been stored on said other disk are written in batch mode onto said other disk without interruption, while said one disk serves said computer system in a read-only mode and writes from said computer system are received into said memory;

means for operating said mirrored pair of disks in said first mode of operation during spaced time intervals and in said second mode of operation during at least a portion of the time between said spaced time intervals; and

means for providing a requested data block to said computer system from said memory if said requested data block is in said memory, and otherwise from said other disk if said mirrored pair of disks is operating in said first mode of operation and from said one disk if said mirrored pair of disks is operating in said second mode of operation,

whereby data blocks are written onto said mirrored pair of disks in sorted order in batched runs without interference from or with the reading of data blocks requested by said computer system.

2. A fault-tolerant disk storage subsystem as defined in claim 1 wherein said controller is implemented by software in said computer system and said memory of said controller is a portion of the general storage resources of said computer system.

3. A fault-tolerant disk storage subsystem as defined in claim 1 wherein said controller and memory are implemented with dedicated hardware.

4. A fault-tolerant disk storage subsystem as defined in claim 1 wherein said memory is non-volatile.

5. A fault-tolerant disk storage subsystem as defined in claim 1 wherein said memory is fault-tolerant.

6. A fault-tolerant disk storage subsystem as defined in claim 1 wherein said means for operating said mirrored pair of disks in said first mode of operation schedules said first mode of operation to start periodically.

7. A fault-tolerant disk storage subsystem as defined in claim 1 wherein said means for sorting said accumulated data blocks sorts said data blocks into an order which corresponds to a physical scan of an entire disk.

8. A fault-tolerant disk storage subsystem as defined in claim 1 wherein said controller includes means for making a single scan through said accumulated data blocks that have not yet been stored on said one disk in sorted order during said first mode of operation before changing modes.

9. A fault-tolerant disk storage subsystem as defined in claim 1 wherein said controller further includes means for providing a third mode of operation during which both of said disks of said mirrored pair are in read mode and a requested data block may be retrieved from either of said disks of said mirrored pair in the event said requested data block is not in said memory.

10. A fault-tolerant disk storage subsystem as defined in claim 1 wherein said controller further includes means for operating said mirrored pair of disks in said third mode of operation whenever said first mode of operation is terminated prior to a next scheduled start of said second mode of operation and whenever said second mode of operation is terminated prior to a next scheduled start of said first mode of operation.

11. A fault-tolerant disk storage subsystem as defined in claim 9 wherein said controller includes means for determining during said third mode of operation, when a data block is requested by said computer system and is not present in said memory, which one of said disks in said mirrored pair can deliver said requested data block in the shortest time and means for retrieving said requested data block from said determined disk.

12. A fault-tolerant disk storage subsystem as defined in claim 1 wherein said controller further includes means for providing a third mode of operation during which both of said disks of said mirrored pair are operated in write mode.

13. A fault-tolerant disk storage subsystem as defined in claim 12 wherein said controller further includes means for operating said mirrored pair of disks in said third mode of operation whenever all accumulated data blocks that have not been stored on said one disk at the start of said first mode of operation have not been written to said one disk by the time said second mode of operation is scheduled to start again and whenever all accumulated data blocks that have not been stored on said other disk at the start of said second mode of operation have not been written to said other disk by the time said first mode of operation is scheduled to start again.

14. A fault-tolerant disk storage subsystem as defined in claim 1 wherein said subsystem includes a spare pair of storage disks and said controller includes means for augmenting said mirrored pair of storage disks with said spare pair of storage disks during a recovery mode of operation in the event that only one of said disks of said mirrored pair remains operational.

15. A fault-tolerant disk storage subsystem as defined in claim 14 wherein said controller includes means for placing said disk of said mirrored pair which remains operational in read-only mode continuously during said recovery mode of operation until all blocks on said remaining operational disk have been transferred either to said memory or to one or both of the disks of said spare pair, and means for replacing said mirrored pair with said spare pair when all blocks on said remaining operational disk of said mirrored pair have been transferred.

16. A method of storing data blocks of digital information received from a computer system in a storage subsystem having a mirrored pair of storage disks and for retrieving data blocks from said storage subsystem upon request from said computer system, comprising the steps of:

temporarily accumulating a group of data blocks received from said computer system in the form of separate writes as batches of data blocks to be stored;

sorting said accumulated data blocks in each batch in an order for efficient batch writing said mirrored pair of disks;

operating said mirrored pair of disks in a first mode of operation in which one disk of said mirrored pair is in write-only mode while the other disk of said mirrored pair is in read-only mode and in a second mode of operation in which said one disk is in read-only mode while said other disk is in write-only mode;

copying onto said one disk during said first mode of operation a batch of accumulated and sorted data blocks in said accumulated group that have not been already written to said one disk;

copying onto said other disk during said second mode of operation a batch of accumulated and sorted data blocks in said accumulated group that have not been already written to said other disk;

operating said mirrored pair of disks in said first mode of operation during spaced time intervals and in said second mode of operation during at least a portion of the time between said spaced time intervals; and

retrieving a data block requested by said computer system from said accumulated group of data blocks if said requested data block is in said accumulated group, and otherwise from said other disk if said mirrored pair of disks is operating in said first mode of operation and from said one disk if said mirrored pair of disks is operating in said second mode of operation,

whereby data blocks are written onto said mirrored pair of disks in sorted order without interference from or with the reading of data blocks requested by said computer system.

17. A method of storing data blocks of digital information as defined in claim **16** and further comprising the step of deleting from said accumulated group any data blocks that have been written to both of said disks of said mirrored pair.

18. A method as defined in claim 17 and further comprising the step of associating first and second flags with each accumulated data block in said group, said first flag associated with any particular data group indicating whether or not said particular data group has been copied to said one disk and said second flag associated with said particular data group indicating whether or not said particular data group has been copied to said other disk.

* * * * *

US005583876A

United States Patent [19]

Kakuta

[11] **Patent Number: 5,583,876**

[45] **Date of Patent: Dec. 10, 1996**

[54] **DISK ARRAY DEVICE AND METHOD OF UPDATING ERROR CORRECTION CODES BY COLLECTIVELY WRITING NEW ERROR CORRECTION CODE AT SEQUENTIALLY ACCESSIBLE LOCATIONS**

[75] Inventor: **Hitoshi Kakuta**, Tokyo, Japan

[73] Assignee: **Hitachi, Ltd.**, Tokyo, Japan

[21] Appl. No.: **317,550**

[22] Filed: **Oct. 4, 1994**

[30] **Foreign Application Priority Data**

Oct. 5, 1993 [JP] Japan 5-273200

[51] **Int. Cl.^{6}** **G11C 29/00**

[52] **U.S. Cl.** **371/40.4**; 371/51.1; 395/441; 395/497.01

[58] **Field of Search** 395/425, 575, 395/275, 482, 497.01, 182.04, 441; 371/51.1, 10.1, 40.4

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,416,915	5/1995	Mattson et al.	395/425
5,418,921	5/1995	Cortney et al.	395/425
5,418,925	5/1995	DeMoss et al.	395/425
5,487,160	1/1996	Bemis	395/441

OTHER PUBLICATIONS

Ousterhout, J. & Douglas, F. Beating the I–O Bottleneck: Case for Log Structured File Systems. Oct. 1988.

Rosenblum, M. & Ousterhout, J. The Design & Implementation of a Log Structured File System. Feb. 1992.

Stodolsky, D. et al. Parity Logging Overcoming the Small Write Problem in Redundant Disk Arrays. May 1993.

Ousterhout, J. & Douglas, F. Log Structured File Systems IEEE Publication; CH2686-4/89/0000/0124.

Primary Examiner—Roy N. Envall, Jr.
Assistant Examiner—Yoncha Kundupoglu
Attorney, Agent, or Firm—Antonelli, Terry, Stout & Kraus

[57] **ABSTRACT**

When new data for writing is sent from a host device, old data and old parities are read after a search time respectively, and a new parity is generated with the new data, the old data and the old parities, and the new parity is stored in a cache memory, and when the number of the new parities corresponding to a plurality of write data becomes more than a predetermined value set by a user or when there is a period of time in which no read request or no write request is issued, new parities are collectively written to a drive for storing parities. In this case, a plurality of new parities are written in a series of storing positions, where a plurality of old parities are stored, in a predetermined access order independent of the stored positions of corresponding old parities. At least to a plurality of storing positions in a track, these new parities are written in the order of positions in a track. To the storing positions which belong to a different track or to a different cylinder, new parities are written in the order of tracks or cylinders.

27 Claims, 14 Drawing Sheets

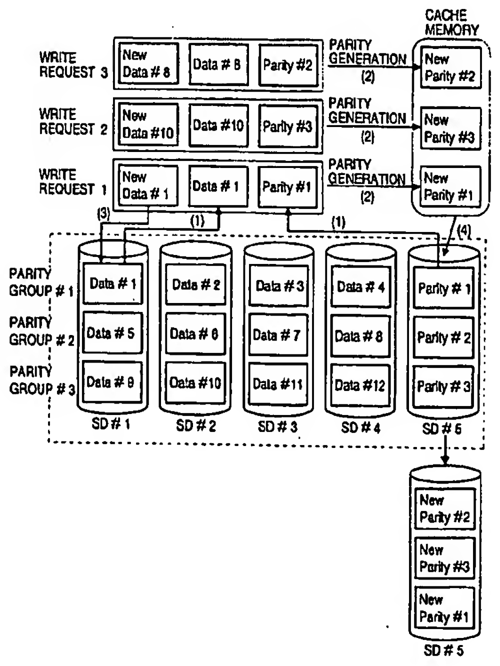
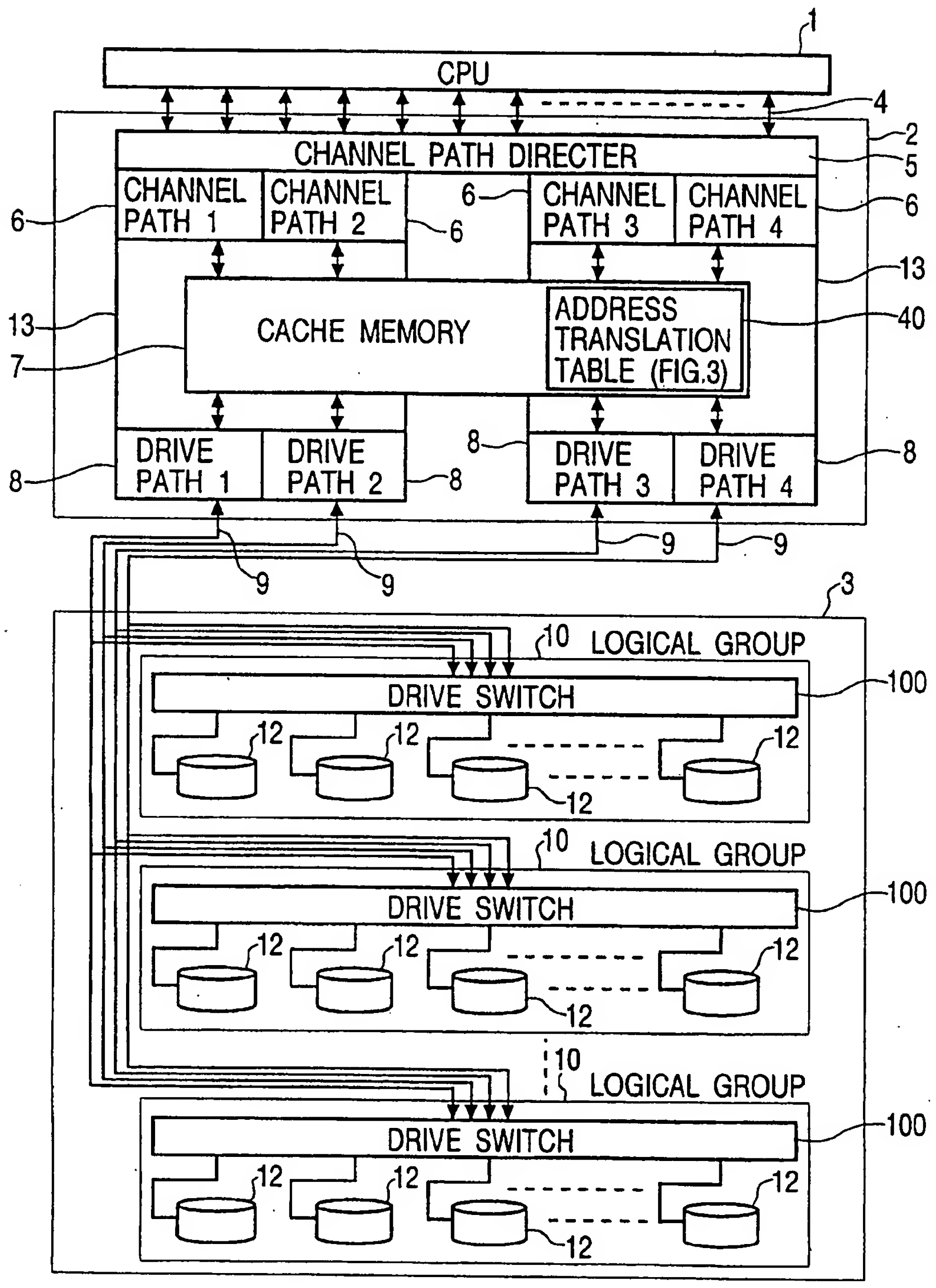


FIG. 1



1
CPU
4
2
CHANNEL PATH DIRECTER
5
CHANNEL PATH 1
CHANNEL PATH 2
CHANNEL PATH 3
CHANNEL PATH 4
6
13
CACHE MEMORY
ADDRESS TRANSLATION TABLE (FIG.3)
40
7
DRIVE PATH 1
DRIVE PATH 2
DRIVE PATH 3
DRIVE PATH 4
8
9
3
10 LOGICAL GROUP
DRIVE SWITCH
100
12

FIG. 2

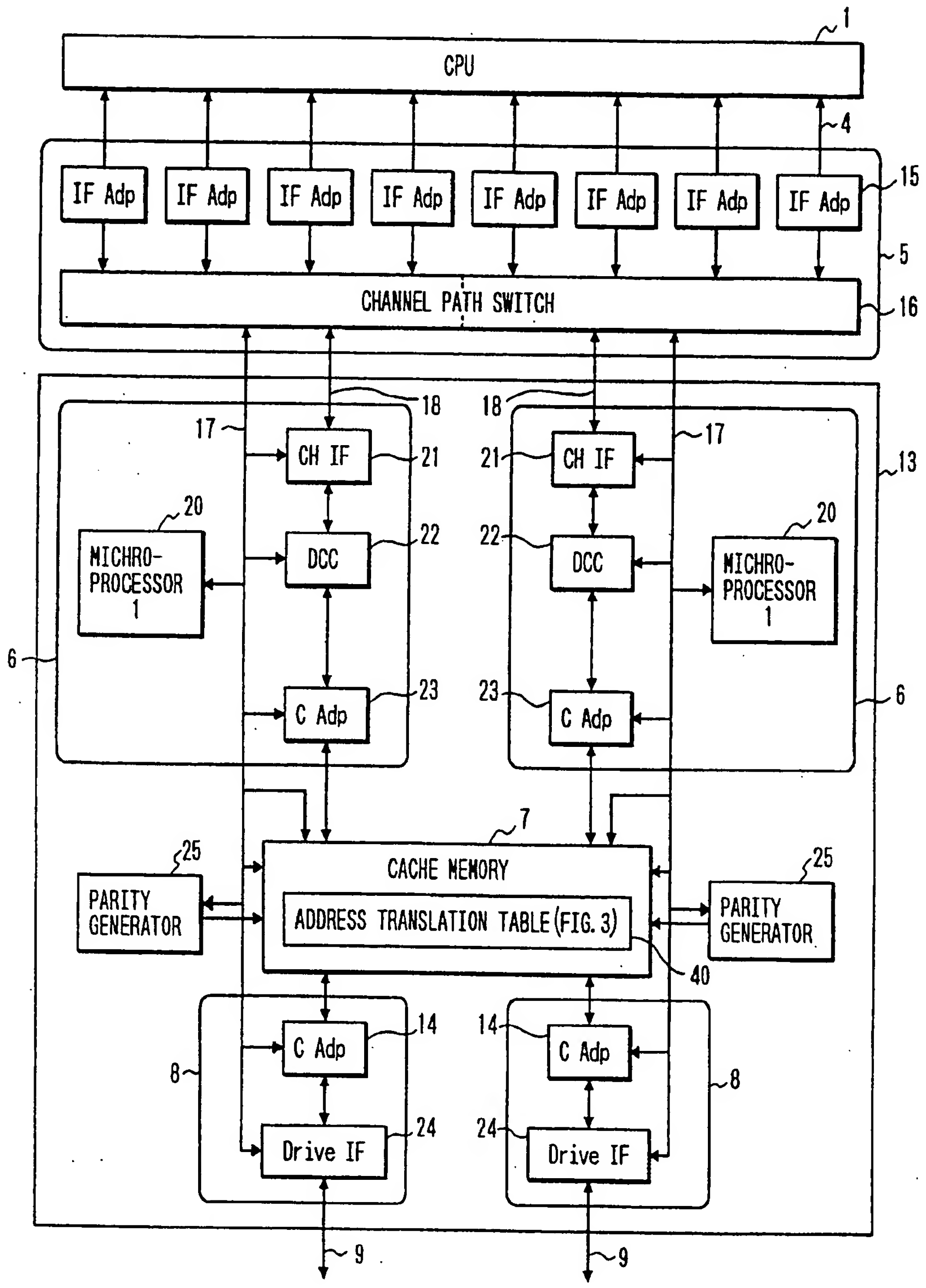

CPU
1
4
IF Adp
15
5
CHANNEL PATH SWITCH
16
18
17
CH IF
21
DCC
22
MICHRO-PROCESSOR 1
20
C Adp
23
6
13
CACHE MEMORY
ADDRESS TRANSLATION TABLE (FIG. 3)
7
40
PARITY GENERATOR
25
C Adp
14
Drive IF
24
8
9

FIG. 3

40

27	28	29	30	31	32	33	34	35	36	37
LOGICAL ADDRESS	INVALIDITY FLAG	DATA DRIVE NO.	INTRA-SCSI ADDRESS	CACHE ADDRESS	CACHE FLAG	PARITY LOGICAL ADDRESS	PARITY DRIVE NO.	PARITY INTRA-SCSI ADDRESS	PARITY CACHE ADDRESS	PARITY CACHE FLAG
Data#1	0	SD#1	DADR1	—	0	Parity#1	SD#5	DADR5	C ADR21	1
Data#2	0	SD#2	DADR1	C ADR5	1					
Data#3	0	SD#3	DADR1	—	0					
Data#4	0	SD#4	DADR1	C ADR2	1					
Data#5	1	SD#1	DADR2	—	0	Parity#2	SD#5	DADR7	—	0
Data#6	0	SD#2	DADR2	C ADR6	1					
Data#7	0	SD#3	DADR2	C ADR1	1					
Data#8	0	SD#4	DADR2	—	0					
......	......	......	......	......	......	......	......	......	......	......

FIG. 4

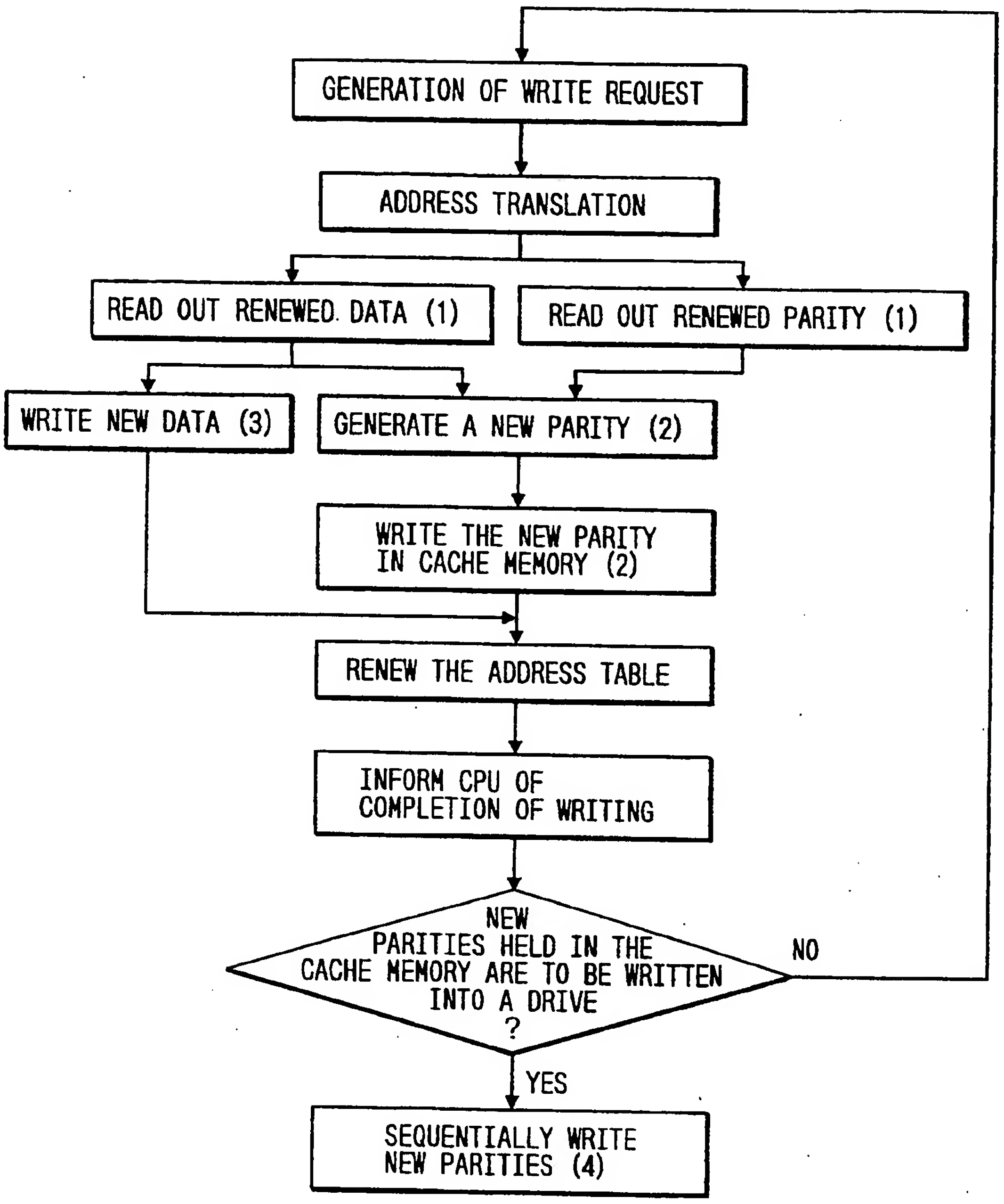

GENERATION OF WRITE REQUEST
ADDRESS TRANSLATION
READ OUT RENEWED DATA (1)
READ OUT RENEWED PARITY (1)
WRITE NEW DATA (3)
GENERATE A NEW PARITY (2)
WRITE THE NEW PARITY IN CACHE MEMORY (2)
RENEW THE ADDRESS TABLE
INFORM CPU OF COMPLETION OF WRITING
NEW PARITIES HELD IN THE CACHE MEMORY ARE TO BE WRITTEN INTO A DRIVE ?
NO
YES
SEQUENTIALLY WRITE NEW PARITIES (4)

FIG. 5

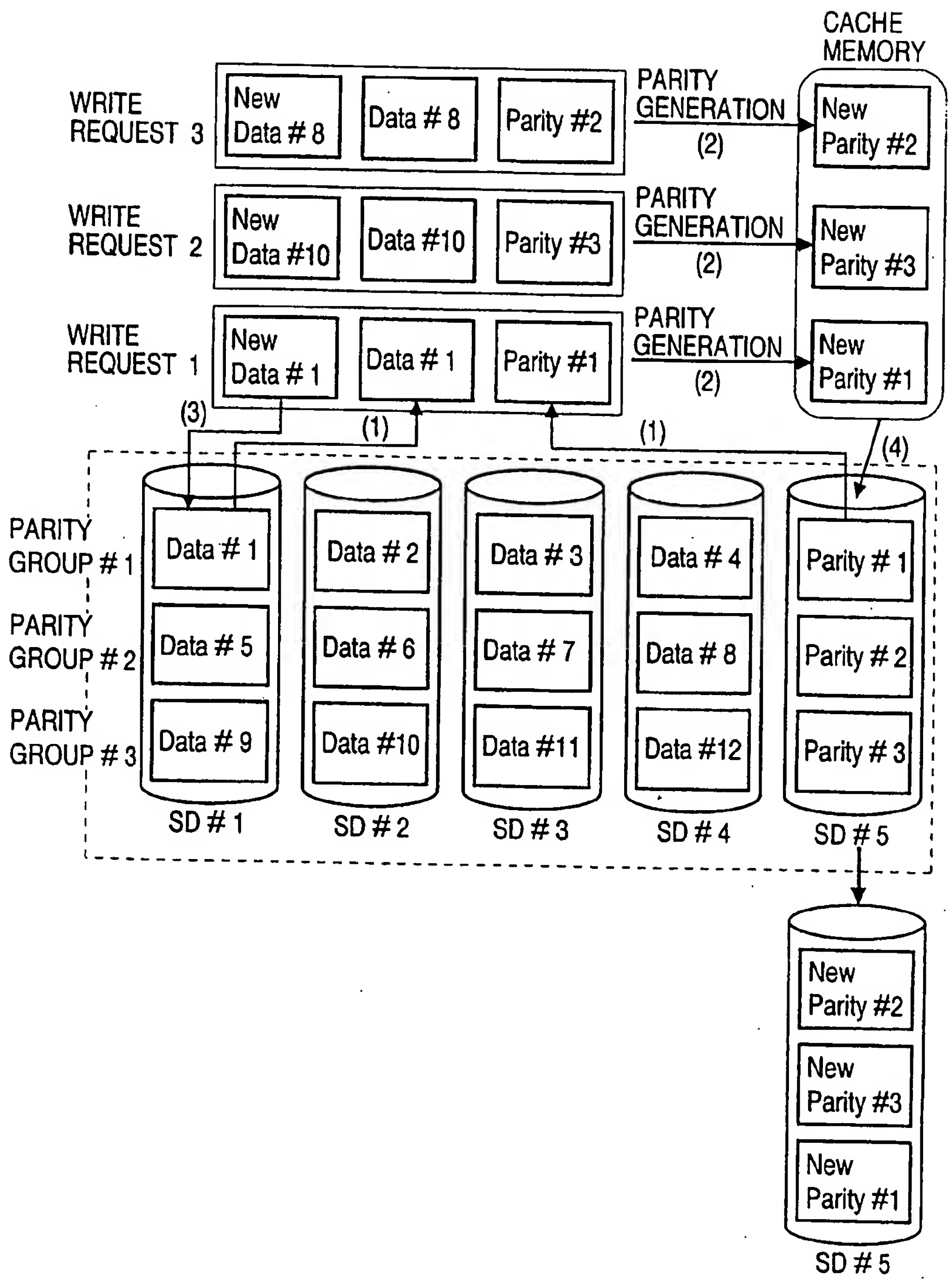

CACHE MEMORY
WRITE REQUEST 3
New Data # 8
Data # 8
Parity #2
PARITY GENERATION
(2)
New Parity #2
WRITE REQUEST 2
New Data #10
Data #10
Parity #3
PARITY GENERATION
(2)
New Parity #3
WRITE REQUEST 1
New Data # 1
Data # 1
Parity #1
PARITY GENERATION
(2)
New Parity #1
(3)
(1)
(1)
(4)
PARITY GROUP # 1
PARITY GROUP # 2
PARITY GROUP # 3
Data # 1
Data # 2
Data # 3
Data # 4
Parity # 1
Data # 5
Data # 6
Data # 7
Data # 8
Parity # 2
Data # 9
Data #10
Data #11
Data #12
Parity # 3
SD # 1
SD # 2
SD # 3
SD # 4
SD # 5
New Parity #2
New Parity #3
New Parity #1
SD # 5

FIG. 6

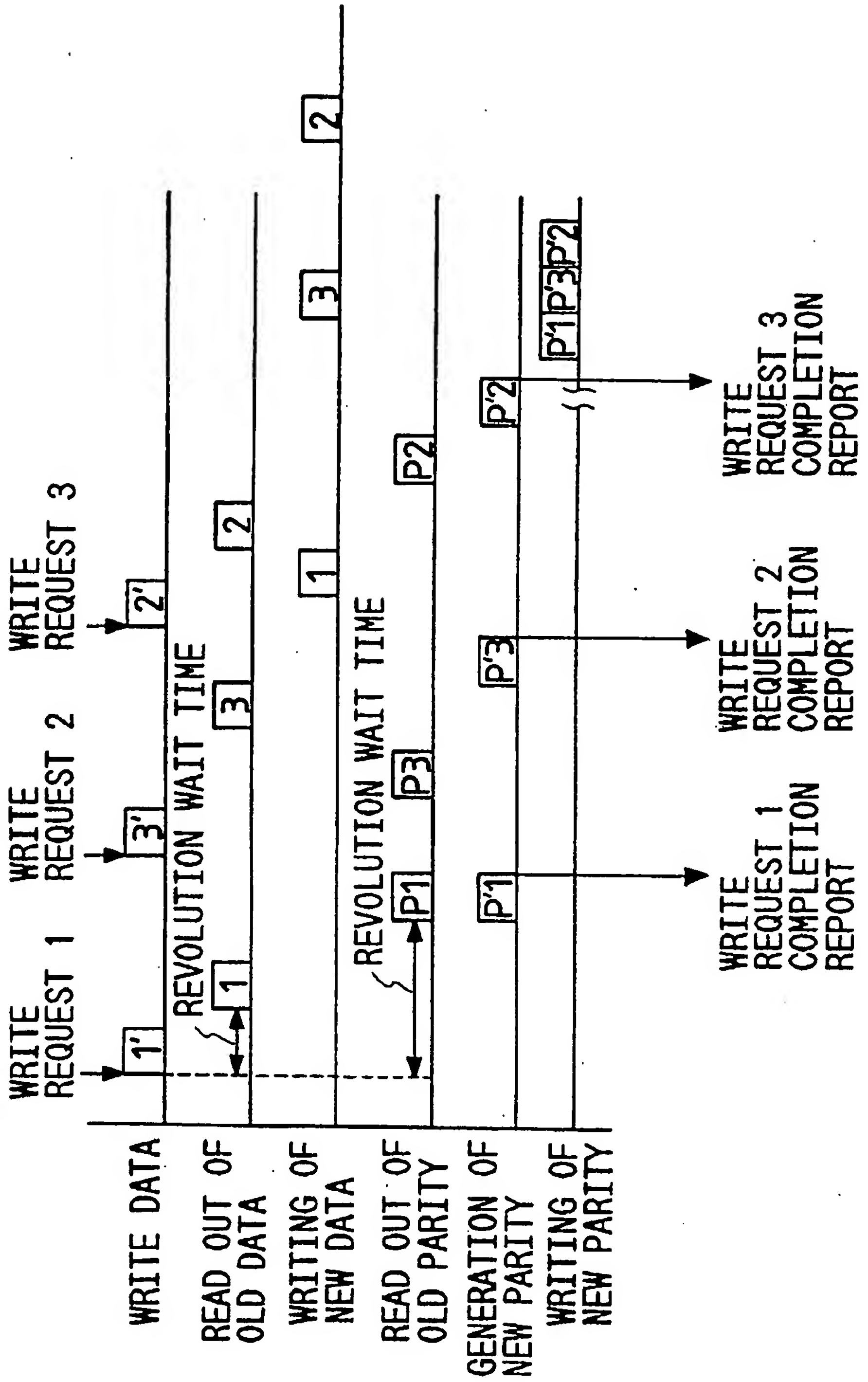

WRITE DATA
READ OUT OF OLD DATA
WRITING OF NEW DATA
READ OUT OF OLD PARITY
GENERATION OF NEW PARITY
WRITING OF NEW PARITY
WRITE REQUEST 1
WRITE REQUEST 2
WRITE REQUEST 3
1'
3'
2'
REVOLUTION WAIT TIME
1
3
2
1
3
2
REVOLUTION WAIT TIME
P1
P3
P2
P'1
P'3
P'2
P'1
P'3
P'2
WRITE REQUEST 1 COMPLETION REPORT
WRITE REQUEST 2 COMPLETION REPORT
WRITE REQUEST 3 COMPLETION REPORT

FIG. 7

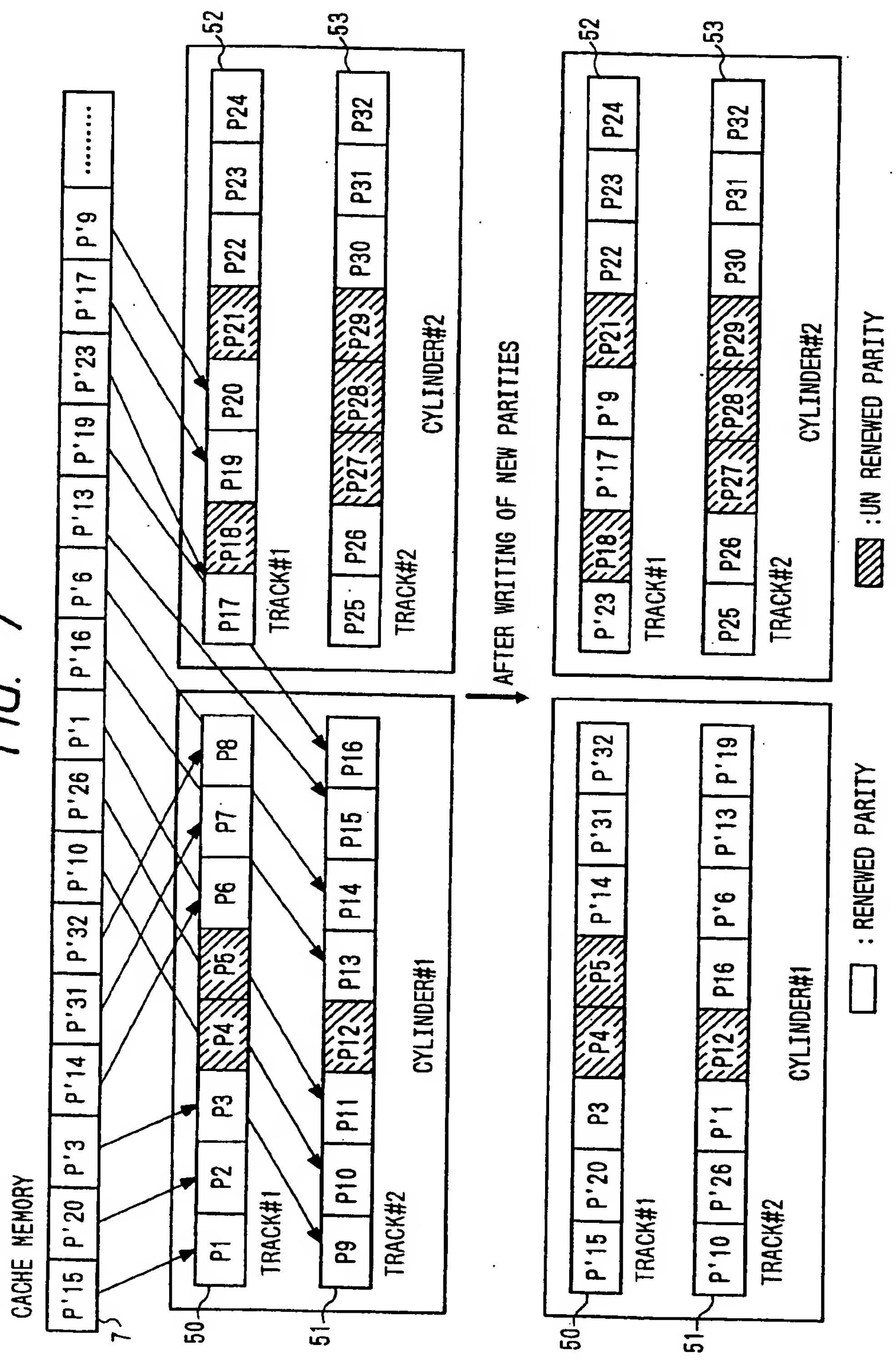
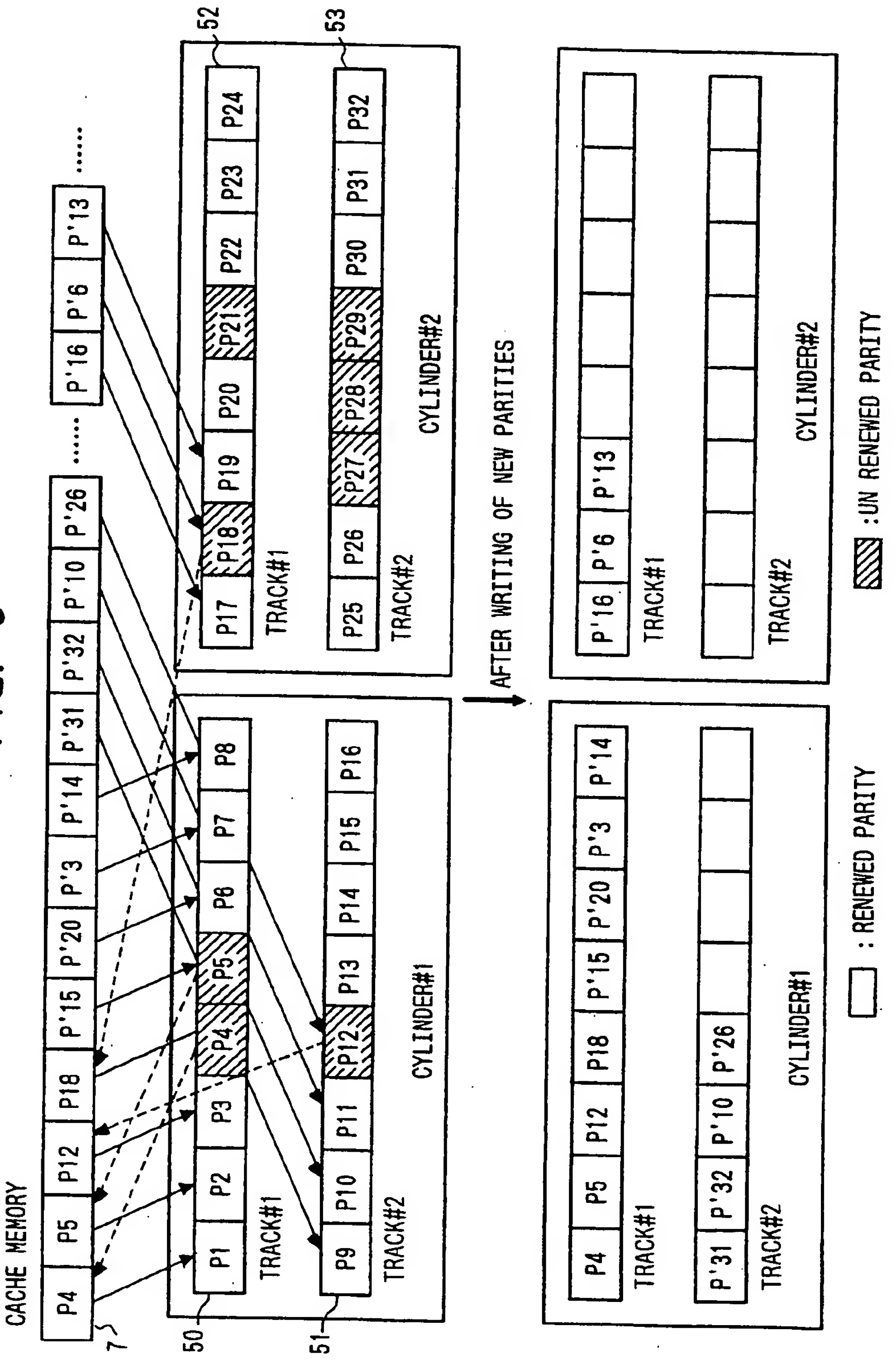

CACHE MEMORY
P'15
P'20
P'3
P'14
P'31
P'32
P'10
P'26
P'1
P'16
P'6
P'13
P'19
P'23
P'17
P'9
....
7
50
51
52
53
P1
P2
P3
P4
P5
P6
P7
P8
TRACK#1
P9
P10
P11
P12
P13
P14
P15
P16
TRACK#2
CYLINDER#1
P17
P18
P19
P20
P21
P22
P23
P24
TRACK#1
P25
P26
P27
P28
P29
P30
P31
P32
TRACK#2
CYLINDER#2
AFTER WRITING OF NEW PARITIES
P'15
P'20
P3
P4
P5
P'14
P'31
P'32
TRACK#1
P'10
P'26
P'1
P12
P16
P'6
P'13
P'19
TRACK#2
CYLINDER#1
P'23
P18
P'17
P'9
P21
P22
P23
P24
TRACK#1
P25
P26
P27
P28
P29
P30
P31
P32
TRACK#2
CYLINDER#2
: RENEWED PARITY
:UN RENEWED PARITY

FIG. 8


CACHE MEMORY
P4
P5
P12
P18
P'15
P'20
P'3
P'14
P'31
P'32
P'10
P'26
......
P'16
P'6
P'13
......
7
50
51
52
53
P1
P2
P3
P4
P5
P6
P7
P8
TRACK#1
P9
P10
P11
P12
P13
P14
P15
P16
TRACK#2
CYLINDER#1
P17
P18
P19
P20
P21
P22
P23
P24
TRACK#1
P25
P26
P27
P28
P29
P30
P31
P32
TRACK#2
CYLINDER#2
AFTER WRITING OF NEW PARITIES
P4
P5
P12
P18
P'15
P'20
P'3
P'14
TRACK#1
P'31
P'32
P'10
P'26
TRACK#2
CYLINDER#1
P'16
P'6
P'13
TRACK#1
TRACK#2
CYLINDER#2
: RENEWED PARITY
:UN RENEWED PARITY

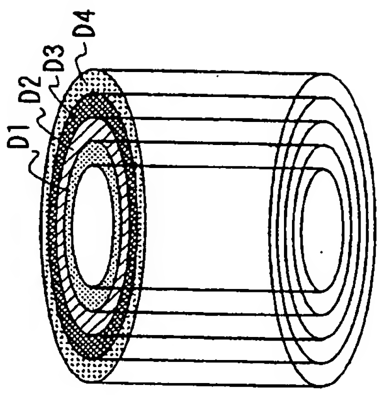


FIG. 9

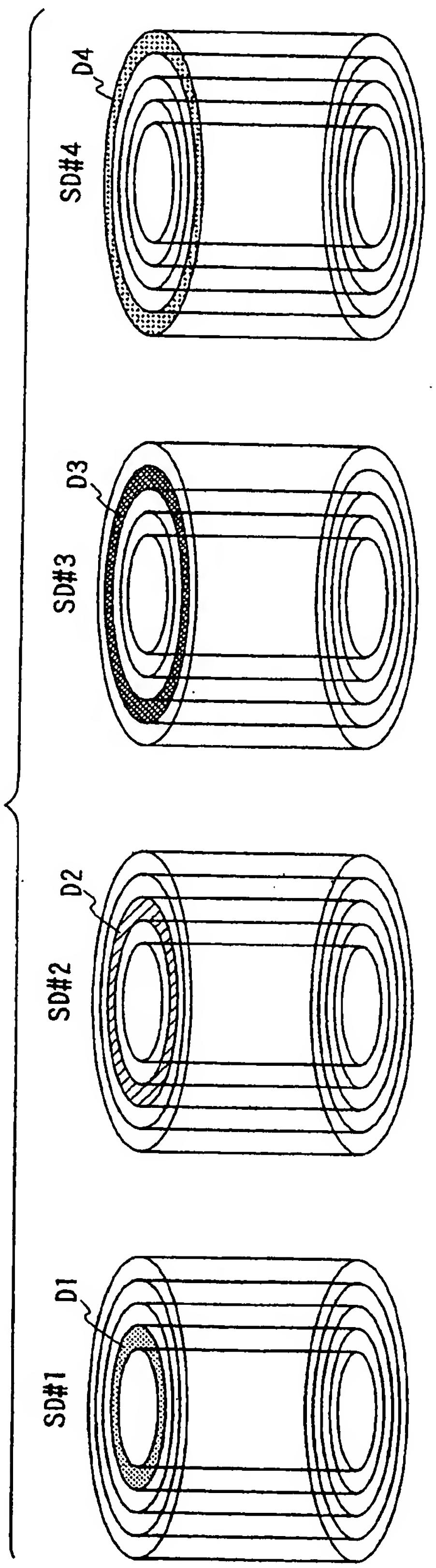


FIG. 10

FIG. 11

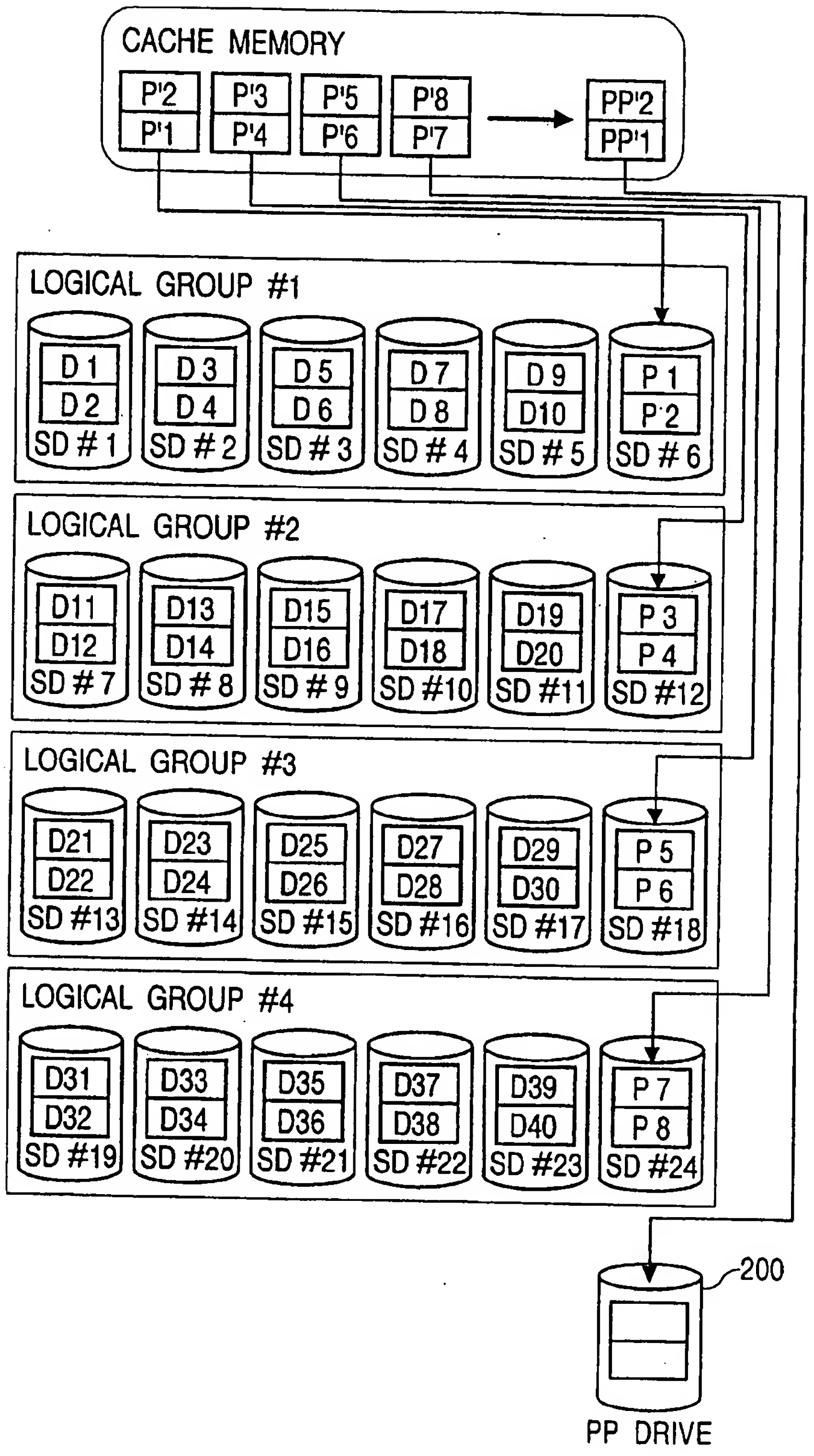

CACHE MEMORY
P'2
P'1
P'3
P'4
P'5
P'6
P'8
P'7
PP'2
PP'1
LOGICAL GROUP #1
D1
D2
SD #1
D3
D4
SD #2
D5
D6
SD #3
D7
D8
SD #4
D9
D10
SD #5
P1
P2
SD #6
LOGICAL GROUP #2
D11
D12
SD #7
D13
D14
SD #8
D15
D16
SD #9
D17
D18
SD #10
D19
D20
SD #11
P3
P4
SD #12
LOGICAL GROUP #3
D21
D22
SD #13
D23
D24
SD #14
D25
D26
SD #15
D27
D28
SD #16
D29
D30
SD #17
P5
P6
SD #18
LOGICAL GROUP #4
D31
D32
SD #19
D33
D34
SD #20
D35
D36
SD #21
D37
D38
SD #22
D39
D40
SD #23
P7
P8
SD #24
200
PP DRIVE

FIG. 12

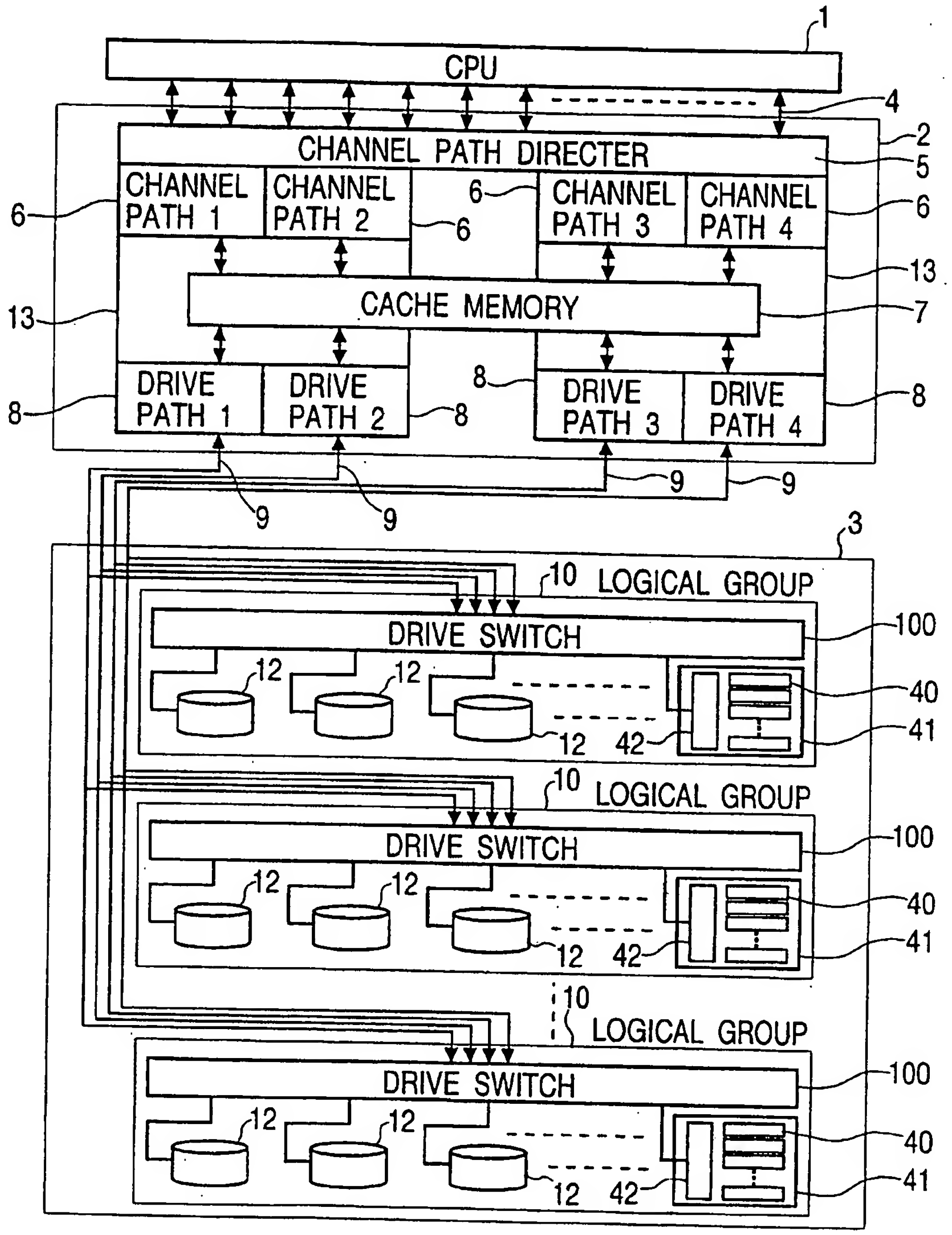

1
CPU
4
2
CHANNEL PATH DIRECTER
5
CHANNEL PATH 1
CHANNEL PATH 2
CHANNEL PATH 3
CHANNEL PATH 4
6
13
CACHE MEMORY
7
DRIVE PATH 1
DRIVE PATH 2
DRIVE PATH 3
DRIVE PATH 4
8
9
3
10 LOGICAL GROUP
DRIVE SWITCH
100
12
40
41
42

FIG. 13

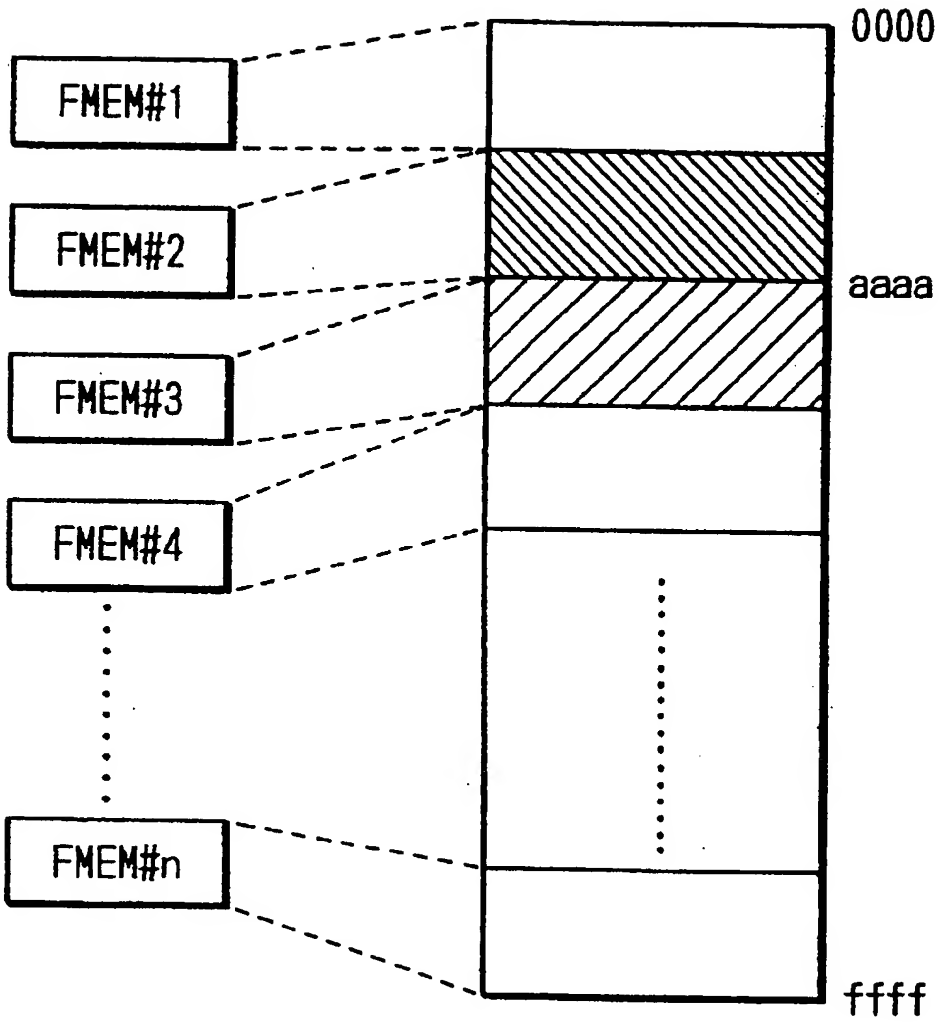


FIG. 14

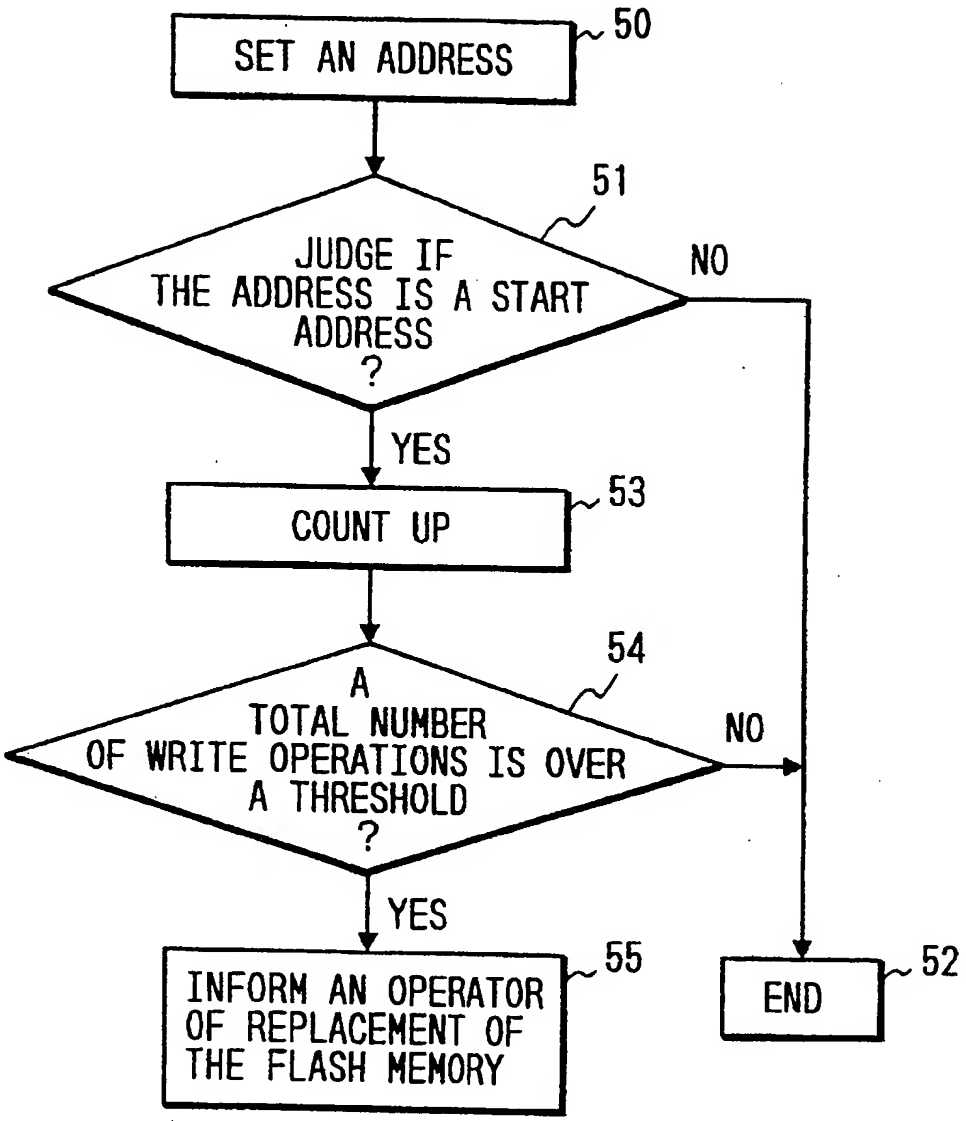

SET AN ADDRESS
50
JUDGE IF THE ADDRESS IS A START ADDRESS ?
51
NO
YES
COUNT UP
53
A TOTAL NUMBER OF WRITE OPERATIONS IS OVER A THRESHOLD ?
54
NO
YES
INFORM AN OPERATOR OF REPLACEMENT OF THE FLASH MEMORY
55
END
52

FIG. 15A
PRIOR ART

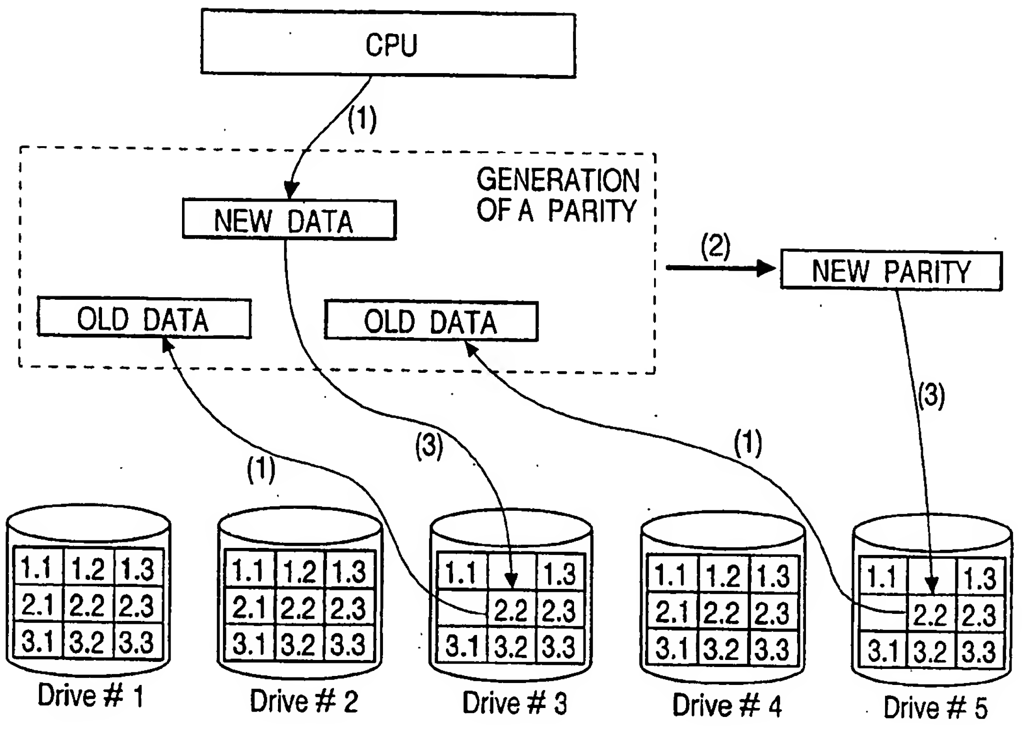
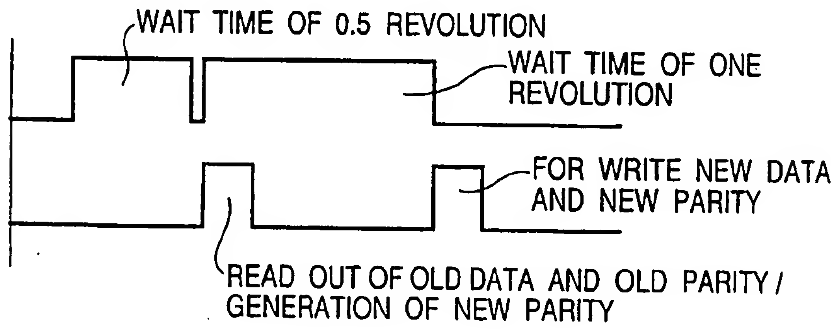


FIG. 15B
PRIOR ART


WAIT TIME OF 0.5 REVOLUTION
WAIT TIME OF ONE REVOLUTION
FOR WRITE NEW DATA AND NEW PARITY
READ OUT OF OLD DATA AND OLD PARITY / GENERATION OF NEW PARITY

DISK ARRAY DEVICE AND METHOD OF UPDATING ERROR CORRECTION CODES BY COLLECTIVELY WRITING NEW ERROR CORRECTION CODE AT SEQUENTIALLY ACCESSIBLE LOCATIONS

BACKGROUND OF THE INVENTION

The present invention relates to a method of updating error correcting codes and a disk array device which is suitable to the updating method, the disk array device comprising a plurality of disks for holding data and a disk for holding error correcting codes for the data held in the plurality of disks.

In a present day computer system, the data needed by a higher order device, such as a CPU, etc., is stored in a secondary storage, and the CPU, as occasion demands, performs read or write operations for the secondary storage. In general, a nonvolatile storage medium is used for the secondary storage, and as a representative storage, a disk device (hereinafter referred to as a disk drive) or a disk in which a magnetic material or a photo-electromagnetic material is used can be cited.

In recent years, with the advancement of computerization, a secondary storage of higher performance has been required. As a solution, there has been proposed a disk array constituted with a plurality of drives of comparatively small capacity. For example, referenced is made to "A Case for Redundant Arrays of Inexpensive Disks (RAID)" by D. Patterson, G. Gibson, and R. H. Kartz read in ACM SIGMOD Conference, Chicago, Ill., (June, 1988) pp 109 to 116.

In a disk array which is composed of a large number of drives, due to the large number of parts, the probability of occurrence of a failure becomes high. Therefore, for the purpose of increasing the reliability, the use of an error correcting code has been proposed. In other words, an error correcting code is generated from a group of data stored in a plurality of data disks, and the error correcting code is written to a different disk than the plurality of data disks. When a failure occurs in any of the data disks, the data in the failed disk is reconstructed using the data stored in the rest of the normal disks and the error correcting code. A group of data to be used for generating the error correcting code is called an error correcting data group. Parity is mostly used as the scheme for an error correcting code. In the following, parity is used for an error correcting code, except for a case under special circumstances; however, it will be apparent that the present invention is effective in a case where an error correcting code other than one based on parity is used. When parity is used for an error correcting code, the error correcting code data group can be also referred to as a parity group.

The above-mentioned document reports the results of a study concerning the performance and the reliability of a disk array (level 3) in which write data is divided and written to a plurality of drives in parallel, and of a disk array (level 4 and 5) in which data is distributed and handled independently.

In a present day large scale general purpose computer system or the like, in a secondary storage, constituted with disk drives, the addresses of individual units of data which are transferred from a CPU are fixed to predetermined addresses, and when the CPU performs reading or writing of the data, the CPU accesses the fixed addresses. The same thing can be said about a disk array.

In the case of a disk array (level 3) in which data is divided and processed in parallel, the fixing of addresses exerts no influence upon the disk array; however, in the case of a disk array (level 4 and 5) in which data is distributed and handled independently, when the addresses are fixed, a data writing process is followed by a large overhead. About the overhead, an explanation has been given in Japanese Patent Application No. Hei 4-230512; in the following also, the overhead will be explained briefly in the case of a disk array (level 4) in which data is distributed and handled independently.

In FIG. 15A, each address (i,j) is an address for a unit of data which can be processed in read/write operation of one access time.

Parity is constituted by a combination of data composed of 4 groups of data in each address (2,2) in the drives from No. 1 to No. 4, and the parity is stored in a corresponding address (2,2) in the drive No. 5 for storing parity. For example, when the data in the address (2,2) in drive No. 3 is to be updated, at first, the old data before the update in the address (2,2) in drive No. 3 and the old parity in the address (2,2) in the parity drive No. 5 are read (1).

An exclusive-OR operation on the read old data, the old parity, and the updated new data is carried out to generate a new parity (2). After the generation of the new parity is completed, the updated new data is stored in the address (2,2) in the drive No. 3, and the new parity is stored in the address (2,2) in the drive No. 5 (3).

In the case of a disk array of level 5, in order to read out the old data and the old parity from the drive on which data is stored and from the drive on which the parities are stored, disk rotation is delayed by ½ turn on the average, and from the disk the old data and the old parity are read out to generate a new parity.

As shown in FIG. 15B, one turn is needed to write the newly generated parity at the address (2,2) in the drive No. 5. A latency time also is needed to write the new data at the address (2,2) in the drive No. 3. In conclusion, for the rewriting of data, at a minimum, a latency time of 1.5 turns is needed. In the case of the RAID 4, since a plurality of parities for the data in a plurality of parity groups are stored on the same disk, a latency time of one turn needed when a new parity is written causes a degrading of the performance in writing. Even if the write time of new data increases, the data access for the data stored on other disks can be performed independently, in principle, so that the influence of the overhead on the write time of data is smaller in comparison with the overhead involving an update of parity.

In order to reduce the overhead during write time as described above, a dynamic address translation method may be employed, as disclosed in PCT International Application laid open under WO 91/20076, applied by Storage Technology Corporation (hereinafter referred to as STK).

In Japanese Patent Application No. Hei 4-230512, applied for by IBM, there is disclosed a method for reducing the write overhead by writing data at an address other than the address at which the write data is to be written.

On the other hand, in recent years, a flash memory has been suggested as a replacement for the magnetic disk. Since a flash memory is a nonvolatile memory, the reading or writing of data in the flash memory can be performed faster in comparison with that in a magnetic disk. In the case of a flash memory, however, when data is to be written, other data existing at the receiving address has to be erased. In the case of a representative flash memory, the write time or the read time is in the order of 100 ns, similar to the case of the RAID, but it takes about 10 ms for an erase time. Also, there is a limit to the number of times writing may be carried out, and the limit is said to be about one million times, which is regarded as a problem in practical use.

In order to solve the above-mentioned problem concerning the limit in the number of times writing is possible in the case of a flash memory, a method in which address translation is performed during the write time so that the number of writing times to flash memories can be averaged with the use of a mapping table is disclosed by IBM in Japanese Patent Application No. Hei 5-27924.

SUMMARY OF THE INVENTION

In the methods described in the above-referenced patent applications by STK and IBM, the overhead at the time of writing data and new parities can be reduced by use of a dynamic address translation method; however, in order to realize such a technique, the processes for the management of space regions (i.e. regions currently not in use) and used regions on the disk must be increased. Therefore, it is desirable to decrease the write overhead using a simpler method.

An object of the present invention is to offer an error correcting code updating method in which the overhead for a writing process in a disk array can be decreased with a simple method.

Another object of the present invention is to decrease the overhead for a writing process of an error correcting code by storing the error correcting code in a flash memory in a disk array.

In order to achieve these objects, in a desirable mode of operation of a disk array according to the present invention, the following operations are performed: a new correcting code is generated for respective requests for writing issued by a host device, the new error correcting codes are temporarily held in a random access memory, and the new error correcting codes are written in a proper order to a disk being used for correcting codes. In this case, individual new error correcting codes are written at the positions in which a plurality of old error correcting codes were held in the positional access order. The access order of a plurality of storing positions in a track is determined according to the order of the positions in the direction of rotation. To the storing positions in the different tracks or cylinders new correcting codes are written in a proper order, for example, in the order of the tracks or in the order of the cylinders.

Thereby, it is possible to write a plurality of new error correcting codes in a short time.

In another desirable mode of operation of a disk array according to the present invention, the following operations are performed: before the groups of new error correcting codes arranged in a sequential order are written to a drive for error correcting codes, a plurality of effective error correcting codes which do not need updating, because no write request is issued for them, are read from the error correcting disk in order, the read error correcting codes which do not need updating are held in a flash memory together with the above-mentioned new error correcting codes, and the series of error correcting codes are written onto the disk for error correcting codes.

In a further desirable mode of operation according to the present invention, a flash memory which has a short access time is used in place of a disk for storing error correcting codes, and a series of new error correcting codes are written to the flash memory according to one of the collective-writing procedures of error correcting codes as described above.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a schematic diagram showing the hardware constitution of the embodiment 1.

FIG. 2 is a schematic block diagram showing the internal constitution of the channel path director (5) and the cluster (13) shown in FIG. 1.

FIG. 3 is a chart showing the details of the address translation table (40) to be used for the device shown in FIG. 1.

FIG. 4 is a flowchart of a data writing process and a parity updating process in the embodiment 1.

FIG. 5 is a conceptional illustrative representation of the data writing process and the parity updating process in the embodiment 1.

FIG. 6 is a timing chart of the writing process in the embodiment 1.

FIG. 7 is an illustrative diagram for explaining a sequential writing method of new parities in the embodiment 1.

FIG. 8 is an illustrative diagram for explaining a sequential writing method of new parities in the embodiment 2.

FIG. 9 is an illustrative diagram for explaining region division in the parity drive in the embodiment 3.

FIG. 10 is an illustrative diagram showing the regions for parities distributed in a plurality of data drives in the embodiment 4.

FIG. 11 is a schematic block diagram for explaining the constitution of a device and the writing process in it in the embodiment 5.

FIG. 12 is a schematic block diagram showing the hardware constitution in the embodiment 6.

FIG. 13 is an illustrative diagram for explaining the address in the flash memory used in the embodiment 8.

FIG. 14 is a flowchart for judging the number of writing times in the flash memory used in the embodiment 8.

FIG. 15A is an illustrative diagram for explaining the procedure of a data updating process in a conventional level 4 disk array.

FIG. 15B is a time chart of a data updating process in a conventional level 4 disk array.

DESCRIPTION OF EMBODIMENTS

(Embodiment 1)

FIG. 1 shows a first embodiment of a disk array system according to the present invention. The constitution itself of the device is known to the public, and the present invention is characterized in the procedure for the collective storage of a plurality of new parities generated for a plurality of data writing requests in a drive for parities, so that the constitution of the device will be explained briefly to the extent necessary for understanding the characteristic features of the invention.

A disk array system is composed of a disk array controller 2 and a disk array unit 3 which are connected to a host device, CPU 1, by way of a path 4. The disk array unit 3 is composed of a plurality of logical groups 10. As explained later, the disk array system in the present embodiment is arranged to be able to access four drives in parallel. Following the above, each logical group 10 in the present embodiment is composed of four drives 12; however, in general, the logical group 10 may be composed of m drives (m: an integer larger than 2). These drives are connected to four disk array unit paths 9 by a drive switch 100. There is no special limit in the number of the drives 12 which constitute each logical group. It is assumed in the present invention that each logical group 10 is a failure reconstruction unit, and a plurality of drives 12 in a logical group 10 are composed of three drives for data and a drive for holding parities generated from the data in the drives.

The disk array controller 2 comprises a channel path director 5, two clusters 13, and a cache memory 7. The cache memory 7 is constituted by semiconductor memories which are made to be nonvolatile by a battery backup, etc. There are stored in the cache memory 7, data to be written to any one of the drives, or data read from any one of the drives, and an address translation table 40 which is used for the access to each of the drives in the disk array. The cache memory 7 and the address translation table 40 are used in common by all clusters 13 in the disk array controller 2.

The cluster 13 provides a gathering of paths which can be operated independently from each other in the disk array controller 2, and each cluster has its own power supply and circuit, which are independent from those of other clusters.

The cluster 13 is constituted with two channel paths 6 and two drive paths 8; there are four channel paths between the channel path director and the cache memory 7, and there are also four drive paths between the cache memory 7 and drives 12. Two channel paths 6 and two drive paths 8 are connected through the cache memory 7.

FIG. 2 shows the internal constitution of the channel path director 5 and a cluster 13.

As shown in FIG. 2, the channel path director 5 is constituted with a plurality of interface adapters 15 which receive the commands from the CPU 1 and a channel path switch 16.

Each of the two channel paths 6 in each cluster 13 is composed of a circuit for transferring commands and a circuit for transferring data. The former is composed of a microprocessor (MP) 20 for processing the commands from the CPU 1, a channel director 5, and a path 17 for transferring commands being connected to the microprocessor 20, etc. The latter is composed of a channel interface circuit (CH IF) 21, which is connected to the channel path switch 16 by line 18, and which takes charge of the interface with the channel director 5, a data control circuit (DCC) 22, which controls the transfer of data, and a cache adapter circuit (C Adp) 23. The cache adapter circuit 23 performs reading of data or writing of data for the cache memory 7 under command of the microprocessor 20, and also it performs monitoring of the condition of the cache memory 7 and exclusive control for the write requests and read requests.

Each of the two drive paths 8 in each cluster 13 is composed of a cache adapter circuit 14 and a drive interface circuit 24. The latter is connected to each logic group 10 through one of the four disk array unit paths 9.

Further in each cluster 13, there are contained two parity generator circuits 25, which are connected to the cache memory 7.

(Address translation table 40)

In the present embodiment, it is assumed that the path 9 is constituted as a SCSI path.

Further, in the present embodiment, a plurality of the drives 12 are used as data drives, except for one drive, in the logic groups 10, and the one drive is exclusively used for a specified parity, such as RAID 4.

Each drive comprises a plurality of disks and a plurality of heads which are provided corresponding to the disks, and tracks which belong to different disks are divided into a plurality of cylinders, and a plurality of tracks which belong to a cylinder are disposed to be accessible in order.

The CPU 1, by means of a read command or a write command, designates the data name as a logical address of the data which is to be dealt with by the command. In the present embodiment, the length of data designated with a write command from the CPU 1 is assumed to have a fixed length. The logical address is translated to an Addr in a physical address SCSI in one of the actual drives 12 by either of the microprocessors 20 in the disk controller 2.

To be more specific, the Addr in the SCSI comprises: the number of a drive in which request data is stored, the cylinder address, the number of a cylinder in the above-mentioned drive 12, a head address, the number of a head which selects a track in a cylinder, and a record address which expresses the position of data in a track.

As shown in the following, a table 40, hereinafter referred to as an address table (FIG. 3), is used for the address translation. The table 40 is stored in the cache memory 7.

The address table 40 holds the address information for individual parity groups held by all logical groups in the disk array. In other words, the address table 40 holds the address information about a plurality of units of data which constitute each parity group and the address information about parities of the parity group as a set.

The address table 40 holds for each unit of data in one of the parity groups: a logical address 27 of the data, a nullifying flag 28 which is turned ON (1) when the data is invalid, the number of a data drive 29 (D Drive No.) in which the data is stored, the Addr 30 in the SCSI which expresses a physical address in the drive in which the data is actually stored, a cache address 31 which expresses the address in the cache memory 7 when the data is present in the cache memory 7, and a cache flag 32 which is turned ON (1) when the data is present in the cache memory 7. In the present embodiment, it is assumed that a plurality of units of data which belong to the same parity group are held in the Addr 30 in the same SCSI in each drive 12 which constitutes one of the logical groups 10.

Further, the table 40 comprises for the parity group: a P logical address 33 which is a logical address of a parity of the parity group, the number of a drive (parity drive) 34 (P Drive No.) in which the parity is stored, an Addr 35 in the PSCSI which is a physical address in the parity drive in which the parity is actually stored, a P cache address 36 which shows the stored position of the parity when the parity is stored in the cache memory 7, and a P cache flag 37 which shows the existence of the parity in the cache memory 7.

When the power supply is turned ON, the address table 40 is read automatically, with no sensing of the CPU 1, into the cache memory 7 from a specified drive in the logical group 10 by either microprocessor 20. On the other hand, when the power supply is turned OFF, the address table 40 stored in the cache memory 7 is automatically stored back into a predetermined position in the specified drive.

(Writing process)

A writing process includes the following: the updating of data in which a user designates a logical address to which the data is to be written and rewrites the data, and new writing of data in a space region. In the present embodiment, for simplification, a writing process for the update of data will be explained with reference to FIG. 4 to FIG. 7.

When a write request is issued from the CPU 1, the following operation is executed under the control of either of the microprocessors 20.

A logical address and new data for writing, which are transferred from the CPU **1**, are stored in the cache memory **7** through a cache adapter circuit **23**. The address in the cache memory at which the data is stored is registered in a cache address **31** in the address table **40** which corresponds to the logical address of the data.

At this time, a cache flag **32** which corresponds to the logical address **27** is turned ON (**1**).

When a further write request is issued from the CPU **1** for the new data held in the cache memory **7**, the new data is rewritten.

Either of the microprocessors **20**, which receives the write request, recognizes drives **12** (designated by the data drive No. field **29** and the parity drive No. field **34** of the address translation table **40** shown in FIG. **3**) in which the data and parities are stored, and Addr **30** in the SCSI and Addr **35** in the PSCSI, the physical address of the drive **12**, from the logical address designated by the CPU **1**, by referring to the address table **40**.

As shown in FIG. **5**, when a write request is issued from the CPU **1** for the Data No. **1** in the drive **12** of SD No. **1** to update the data to a New Data No. **1**, the microprocessor **20**, after recognizing the physical addresses of Data No. **1**, the data to be updated (old data), and of parity No. **1**, the parity to be updated (old parity), by referring to the address table **40**, reads out the old data and the old parity from respective drives (step **1** in FIG. **4** and FIG. **5**). The read out old data and old parity are stored in the cache memory **7**.

An exclusive-OR operation is performed on the old data and the old parity, and new data to be written, to generate a New Parity No. **1**, a new parity after updating, and the new parity is stored in the cache memory **7** (step (**2**) in FIGS. **4** and **5**).

After the completion of storing of the new parity (New Parity No. **1**) into the cache memory **7**, the microprocessor **20** writes new data (New Data No. **1**) in the address of Data No. **1** in the drive **12** in SD No. **1** (step (**3**) in FIGS. **4** and **5**). The writing of the new data can be performed asynchronously under the control of the microprocessor **20**.

The characteristic of the present invention is that a new parity (New Parity No. **1**) is not immediately written in the parity drive SD No. **5**, but is put together with other new parities corresponding to other write requests and they are written en bloc in the parity drive.

In the case where New Data No. **1** is registered in an entry having a logical address of Data No. **1** for the address table **40** and the data is held in the cache memory **7**, the address in the cache memory **7** is registered in the cache address **31**, and the cache flag **32** is turned ON. About parity, the cache address is registered in the Pcache address **36**, and the Pcache flag **37** is turned ON.

As described above, in the address table **40**, a parity whose Pcache flag is in the ON state is classified to be an updated parity, and a parity stored in a drive for storing parity is regarded to be invalid thereafter.

In the present embodiment, as shown in FIG. **6**, when the new data and the new parity from the CPU **1** are stored in the cache memory **7**, the microprocessor **20** reports to the CPU **1** that the process of writing the data is finished.

In a conventional method, after a new parity is written in a parity drive with a delay of one turn, the microprocessor **20** reports to the CPU **1** that the writing process is finished.

The writing of a new parity into the parity drive No. **5** is performed asynchronously under the control of the microprocessor **20**, so that it cannot be seen by a user.

When the writing of a new data into the data drive No. **1** is performed asynchronously under the control of the microprocessor **20**, the completion of the writing is not seen by a user.

Following the above, when a write request for other data, for example, Data No. **10** and Data No. **8** stored in data drives SD No. **2** and SD No. **4** is issued, these requests are processed in the same way as described above, and new data corresponding to the above-mentioned data is written in a data drive and new parities corresponding to them are stored in the cache memory **7**.

When a number of the new parities collected in the cache memory **7** exceeds a preset value set by a user, or when there is a period of time in which no read/write request is issued from the CPU **1**, these new parities are written collectively in order into a parity drive SD No. **5** using a method to be explained later (step (**4**) in FIGS. **3** and **4**).

After such writing process is finished, Addr in SCSI in which new parities are actually written are registered in the Addr **35** in the PSCSI in the address table **40**.

(Collective writing of a plurality of new parities)

In the present embodiment, a plurality of new parities held in the cache memory **7** are written into a parity drive in a sequential order. The positions in the parity drive into which individual new parities are written are different from the case of a conventional technique, wherein new parities are written in the positions where the old parities corresponding to new parities were held. In this regard, in the present embodiment, new parities are written into the positions determined by the order of access among a plurality of storage positions in which a plurality of old parities were held.

The order of access is determined such that a plurality of storing positions which belong to a track are accessed in the order of the storage positions in the direction of rotation.

In the present embodiment, a plurality of new parities held in the cache memory are written into a parity drive in the order of occurrence of the write requests which have caused the new parities to be generated.

To be more specific, a detailed explanation will be given in the following.

(1) In finding a plurality of tracks to which the old parities corresponding to the new parities belong, a group of the new parities held in the cache memory **7** are written in order into the positions of old parities according to the predetermined cylinder order and track order. In such case, to the positions of a plurality of old parities in a track, new parities are written in order following the order of the storage positions in the track.

For example, in FIG. **7**, there is shown a state where a group of old parities in the track **50** (No. **1**) and track **51** (No. **2**) in the cylinder No. **1**, and in the track **52** (No. **1**) and track **53** (No. **2**) in the cylinder No. **2** are rewritten to a group of new parities in the cache memory **7**. In this case; it is assumed that a new parity group,

P'15, P'20, P'3, P'14,P'31, P'32,
P'10, P'26, P'1, P'16, P'6, P'13, P'19,
P'23, P'17, P'9, P'22, P'23, P'24,
P'25, P'26, P'30, P'31, P'32,

is generated in this order, and a group of old parities corresponding to the new parities belong to tracks No. **1** and No. **2** in each of the cylinders No. **1** and No. **2**.

These new parities are written in order from the top into the position of an old parity in the track No. **1** in the cylinder

No. 1, and then the new parities are written into the positions of old parities in the order of storage positions in the track. In this case, new parities, P'15, P'20, P'3, P'14, P'31, and P'32, are written in the positions of old parities, P1, P2, P3, P6, P7, and P8. It is possible to finish the writing process as described in the above in a single turn of a parity drive.

(2) A part of the rest of the new parities are written in a predetermined next track in the same cylinder No. 1, in this case, they are written in the track No. 2.

In the example shown in FIG. 7, new parities, P'10, P'26, P'1, P'16, P'6, P'13, and P'19, are written in the track No. 2 in a sequential order.

(3) A part of the rest of the new parities are written in the next cylinder, in this example, in a track in the No. 2 cylinder, in this case, at a position in the track No. 1.

In the example shown in FIG. 7, new parities P'23, P'17, P'9, P'22, P'23, and P'24, are written in the track No. 1 in order.

(4) A part of the rest of the new parities are written in the same way into the next track in the same cylinder, in this case, in the track No. 2.

In the example shown in FIG. 7, new parities, P'25, P'26, P'30, P'31, and P'32, are written in the track No. 2 in order.

In the way described above, a group of new parities can be written in a short search time into a group of storing positions to which a group of old parities belong.

In a conventional method, the Addr 35 in the PSCSI for storing parities in a parity drive is made to be identical with the Addr 30 in the SCSI for storing data in the drive 12 for storing data. In the present embodiment, however, a plurality of new parities are written in different positions from the storing positions of the corresponding old parities, that is, they are written in the storing positions of the old parities in the access order of the positions; thereby a plurality of new parities can be written in a short time.

As described above, when parities are written sequentially, after a parity has been written, in order to prevent a block which is necessary for writing the next parity to be passed by while a process to start the writing of the next parity is performed, it is necessary to have enough of a sector gap available (On this point, refer to, for example, "Transistor Technology, Special No. 27" CQ Publishing Co., 1, May, 1991, p 20.).

In a case where a new write request is issued from the CPU 1 while new parities collected in the cache memory 7 are being written sequentially, collectively, the above-mentioned writing process is continued and after the writing process is finished, the new write request is processed.

(Reading process)

The reading process is basically the same as the method known to the public. When a read request issued from the C2U 1 is processed, similar to the case where a write request is processed, a logical address designated by the request is translated to a physical address and the desired data is read from one of the data drives, and the read data is transferred to the CPU 1. Provided that, when the data is held in the cache memory 7, the data is transferred from the cache memory 7 to the CPU 1.

(Failure reconstruction process)

When a failure occurs in one of the drives, a method for reconstructing the data in the failed drive will be explained.

For example, as shown in FIG. 5, it is assumed that a Parity No. 1 in a drive 12 in SD No. 5 is composed of Data No. 1 in a drive 12 in SD No. 1, Data No. 2 in a drive 12 in SD No. 2, Data No. 3 in a drive 12 in SD No. 3, and Data No. 4 in a drive 12 in SD No. 4. In the same way, it is assumed that Parity No. 2 is composed of Data Nos. 5, 6, 7 and 8, and Parity No. 3 is composed of Data Nos. 9, 10, 11 and 12.

When a failure occurs at any one of the drives SD Nos. 1, 2, 3 and 4, for example, at SD No. 1, all the data in the failed drive SD No. 1 can be reconstructed from the data in the rest of the drives, SD No. 2. SD No. 3, and SD No. 4 and a parity in the drive No. 5.

In accordance with the present invention, in the result of the collective writing of the plurality of parities, in general, the addresses in the drives SD No. 1 to SD No. 4 of a plurality of units of data which belong to the same parity group do not coincide with the addresses of parities in the parity group in the parity drive SD No. 5.

Therefore, when the data in the failed drive SD No. 1 is to be reconstructed, either of the microprocessors 20 reads out all parities in the parity drive SD No. 5 in the cache memory 7. In that case, the microprocessor 20 registers the addresses in the cache memory 7, which stores the read out parities to the Pcache address 36 in the address table 40, and turns the Pcache flag 37 ON.

Next, the microprocessor 20 reads out three units of normal data which belong to one of the parity groups in drives 12 in SD Nos. 2, 3 and 4, for example, Data No. 2, Data No. 3 and Data No. 4, in order and searches the cache address for a parity which belongs to the same parity group with these units of data from the address table 40. A parity designated by the located address, and the above-mentioned three units of normal data are sent to a parity generating circuit (PG) 25 (FIG. 1) and a unit of data in the failed drive SD No. 1 which belongs to the parity group, for example, Data No. 1 is reconstructed. In the same way, other units of data in the failed drive SD No. 1, for example, Data Nos. 5, 9, etc. are reconstructed.

After the failed drive SD No. 1 is replaced by a normal drive, the reconstructed data is written to the normal drive; thus, the reconstruction process is completed.

In a case where a standby normal drive is provided beforehand for the occurrence of a failure in the drives Nos. 1 to 5, the reconstructed data is written in the standby normal drive.

In the present embodiment, data, except for parity, which belongs to one parity group is held at the same address in different data drives, but the address of a parity in a parity drive differs from that in the former. When a failure occurs in one of the data drives, the parity data which corresponds to the data in a different parity group read from respective data drives in order have to be read from a random position of a parity drive. In the present embodiment, in place of a random reading, it is arranged that all parities are read from parity drives and stored in the cache memory 7, and the parities which correspond to respective parity groups read from a plurality of data drives are utilized by retrieving them from the cache memory 7. Thereby, when a failure is to be reconstructed, the utilization of parities can be performed at a high speed.

(Modification of Embodiment 1)

In the above explanation, the cache memory 7 which stores updated new parities is assumed to be a nonvolatile semiconductor memory. However, a parity, different from data, can be reconstructed even if it is erased by an interruption of electricity or other factors, so that if the overhead for regenerating parity is permitted, it is possible to constitute a region for storing old parities in the cache memory 7 with a volatile semiconductor memory.

In the above explanation, updated new parities are stored in the cache memory 7; however, it is also possible to provide an exclusively used memory for storing them.

(Embodiment 2)

There is another method of collective writing of a plurality of new parities in accordance with the present invention. As shown in FIG. 8, effective parities P8 and P9, which are not to be updated by a write request, are read in the cache memory 7 in response to an instruction from the microprocessor 20, and according to the read parity, the microprocessor 20 sets the Pcache address 36 in the address table 40 and turns the Pcache flag 37 ON; thereby, the above-mentioned effective parities are regarded to be new parities to be updated, and they can be put together with the other new parities for sequential, collective-writing.

For example, as shown in FIG. 8, a group of new parities, as explained with reference to FIG. 7, are held in the cache memory 7, and after that old parities P4, P5, P12, P18, etc. for which there are no write requests and for which originally there was no need for an update, are regarded to be a parity group to be updated and they are held in the cache memory 7. The parity group which is held in the cache memory 7, as described above is written in order into a track to which the new parities belong. In the example shown in FIG. 8, following the parity groups P4, P5, P12, and so on, which did not need rewriting originally, there are generated new parity groups, P'15 and so on, which are written in order into the track No. 1 and track No. 2 in the cylinder No. 1 and then into the track No. 1 and track No. 2 in the next cylinder No. 2.

In the embodiment 2, there is a need for a surplus process in that the parities which do not need updating originally have to be read from a drive; however, as opposed to the embodiment 1, the read out parities and the generated new parities can be written collectively into respective tracks in order, so that the write control becomes simpler than that in the embodiment 1. The method shown in the present embodiment, in the same way as the method in the embodiment 1, can be applied to any of the following embodiments.

(Embodiment 3)

In the present embodiment, as shown in FIG. 9, a parity drive is divided into a plurality of regions, and the collective-writing of parities as described in the above is performed by the units of regions.

The dividing of a region is performed by the Addr 30 in the SCSI.

For example, the Addr 30 in the SCSI designates a plurality of cylinders which belong to a group of cylinders, from one cylinder to another cylinder, to be a region D1.

In the drives 12 in the SD Nos. 1, 2, 3 and 4, the Addr 30 in the SCSI designates that the parity corresponding to the parity groups which belong to these cylinders, that is, the Addr 36 in the PSCSI of the drive 12 in the SD No. 5, is stored in a region to which these cylinders belong.

As described above, in the address table 40, the region to which the parity belongs and the region to which the data belong are made to correspond to each other.

In a case where such dividing of a region is executed, when the updating of a parity which belongs to the region D1 is performed by a data write request, the new parity is held in the cache memory 7 as a parity in the region D1.

In the same way, new parities generated by other write requests from the CPU 1 are held in the cache memory 7, and the new parities which are held as parities in the region D1 are written into the region D1 sequentially, and collectively.

For the other regions D2, D3 and D4, in the same way as described above, new parities which belong to respective regions are written sequentially, and collectively, into respective regions using the method described with reference to the embodiments 1 or 2.

In the present embodiment, when reconstruction of data for a failed drive is to be performed, the same process as that in the embodiment 1 can be performed in each parity group to which parities in each parity region belong. In other words, all parities in each parity region are read in the cache memory 7, and then the reconstruction of data in the parity group to which the parities belong can be performed. The same process will be performed in order for different regions.

As a result, in the present embodiment, when a failure is reconstructed, the total quantity of parities to be read in the cache memory 7 is smaller than those in the embodiment 1.

(Embodiment 4)

In the present embodiment, as shown in FIG. 10, the function of a drive for the regions D1, D2, D3 and D4 to which parities are written is not limited to the function of one drive which is used exclusively for parities, but the function can be distributed to a plurality of drives SD No. 1 to SD No. 4, which constitute a logical group 10. The other points except the above are the same as in the case of the embodiment 3.

(Embodiment 5)

In the present embodiment, when a failure occurs in any of the parity drives, it is possible to expedite the reconstruction of a parity in the failed parity drive utilizing the collective-writing described with reference to the embodiment 1.

In other words, the collective-writing of parities described in the embodiment 1 is executed in parallel in a plurality of parity drives contained in different logical groups, and further, in parallel to the above operation, a different parity is generated from the parities in these parity drives and is written into the other parity drive, which is provided in common to the other parity drives. When a failure occurs in any of the plurality of parity drives, the parity in the failed parity drive is reconstructed using the new parity and normal parities in the parity drives. Heretofore, to reconstruct a failed parity drive, data has been read from all data drives in a logical group to which the parity drive belongs, and new parities have been generated from them. In the present embodiment, a failed parity drive can be reconstructed faster than in the conventional case.

To be more specific, in the logical group No. 1 shown in FIG. 9, updated new parities P'1, P'2, . . . , corresponding to parities P1, P2, . . . , are held in the cache memory 7 in this order using the method described with reference to the embodiment 1. Similar to the above, in the logical group No. 2, updated new parities P'4, P'3, . . . , corresponding to parities P3, P4, . . . , are held in this order in the cache memory 7. Similar to the above, in the logical group No. 3, updated new parities P'6, P'5, . . . , corresponding to parities P6, P5, . . . , are held in the cache memory 7 in this order. Similar to the above, in the logical group No. 4, updated new parities P'7, P'8, . . . , corresponding to parities, P7, P8, . . . , are held in the cache memory 7 in this order. These new parities are written in order to respective drives SD No. 6, SD No. 12, SD No. 18 and SD No. 24 similar to the case of the embodiment 1.

In the present embodiment, a new parity PP'1 is generated from the new parities P'1, P'4, P'6 and P'7 in the logical groups No. 1 to No. 4. Similar to the above, new parity PP'2 is generated from P'2, P'3, P'5 and P'8. These new parities are stored in another drive 200 which is provided in common to these logical groups. When a failure occurs in any of the parity drives SD No. 6, SD No. 12, SD No. 18 or SD No. 24, a parity in the failed drive is found from the rest of the normal drives and a common drive 200; thus, parities in the failed drive are regenerated.

(Embodiment 6)

FIG. 12 shows the hardware constitution of the present embodiment, the same symbols or reference numerals as those in FIG. 1 designate the same elements. The differences in the device shown in FIG. 12 from the device shown in FIG. 1 are: in place of a drive for holding parities, as shown in FIG. 1, a flash memory (FMEM) 41 is provided as a memory for holding parities in logical groups 10. The flash memory 41 is composed of a flash memory controller (FMEMC) 42 and a plurality of flash memory chips 40. In addition to this, in the present embodiment, in place of a parity drive No., P Drive No., and the Addr 35 in the address P SCSI in a drive, a flash memory No. and an Addr in the flash memory are used.

Only the different points between the present embodiment and the embodiment 1 will be explained.

In the present embodiment, the process of writing or reading data is the same as that in the embodiment 1. The collective-writing of parities in this embodiment differs from that in the embodiment 1 in that an erasing of the flash memory 41 is employed.

Before new parities are written collectively, and sequentially, into the flash memory 41, either of the microprocessors 20 examines the quantity of the new parities to be written, and erases the addresses corresponding to the examined quantity in the flash memory chips at one time in which invalid parities (P1, P3 and P2) are written. After that, similar to the embodiment 1, new parities are written collectively, and sequentially.

In the case of a flash memory, when new data is written into the flash memory, at first old data stored in the address at which new data is to be written is erased, and after the erase process is completed, the new data is actually written. In the case of a flash memory, it takes the same period of time to erase one sector (When a flash memory is accessed, the address of the same format is used as the address when a disk is accessed.) and to erase a plurality of sectors at one time. A greater part of a write time is occupied by an erase time, and an actual write time to a flash memory is negligibly small in comparison with the erase time. Owing to the sequential, collective writing, a characteristic of the present embodiment, an erase process can be completed at one time, so that the greater the number of collected new parities is, the smaller the overhead can be made.

(Embodiment 7)

In a device described in the embodiment 6, it is also possible to execute collective-writing of parities by the method shown in the embodiment 2.

In this case, similar to the embodiment 2, parities other than the parities to be updated are also read out from the flash memory 41, and together with the generated new parities they are written collectively to the memory. In the present embodiment, as opposed to the embodiment 2, in a period of time between the read out and the collective writing, an erase operation is executed at one time for the positions of old parities to be updated, which are stored in the flash memory, and for the positions in which the above-mentioned read out parities are held.

(Embodiment 8)

A flash memory, in comparison with a disk drive, has the merit that even if it is accessed at random, the access can be processed in a short time; on the other hand, the number of writing times has a limit. Therefore, if writing is concentrated to a single address, a part of the flash memory may reach the limit of the number of times it is capable of being written. After that, writing to the flash memory may become impossible.

In the present embodiment, in order to reduce the problem, when new parities are sequentially, collectively written to a plurality of flash memory chips 40 which constitute the flash memory 41, a chip to be written is changed in a regular order to average the number of times of writing to all flash memory chips 40.

To be specific, as shown in FIG. 13, it is assumed that the flash memory 41 is composed of n pieces of flash memory chips, Nos. 1, 2, 3, 4, . . . , n, and addresses are from 0000 through ffff. When either of the microprocessor 20 recognizes that the sequential, collective-writing of new parities is to be performed, it examines the last address among the addresses in which new parities are stored, when the collective-writing was performed in the preceding writing times.

For example, in the collective-writing in the preceding times, assuming that new parities are written to the addresses from 0000 through aaaa in the flash memory 41, the microprocessor 20 stores the address aaaa. When the next sequential, collective-writing is to be performed, the microprocessor 20 examines the stored address (aaaa). As described above, if the microprocessor stores the last address used in the preceding writing times, it judges that the next sequential collective-writing of new parities is to be started from the address next to the address aaaa.

When the microprocessor 20 recognizes the head address of the sequential collective-writing of new parities as described above, in the next step, it judges if the number of times of writing to the flash memory 41 reaches the limit.

As shown in the judgment flowchart for the number of writing times in FIG. 14, the microprocessor 20 determines whether the head address of a sequential, collective-writing of new parities is 0000 (51). If it is not 0000, the judgment flow is finished (52), and if it is 0000, the microprocessor 20 adds 1 to the counter value of the number of writing times counter (53). In other words, every time a head address comes, the counter value is increased.

Next, the microprocessor 20 judges if the counter value plus 1 corresponds to the preset limiting value of the number of writing times for a flash memory chip 40 (54). The limiting value of the number of times of writing for a flash memory chips 40 is set by a user for the microprocessor 20 during initialization.

In the judgment result, when the number of writing times to the flash memory chip 40 does not exceed the limiting value, the judgment flow is finished (52), and when the number of writing times exceeds the limiting value, the microprocessor 20 indicates the need for flash memory chip (55).

As described above, in the present embodiment, sequential collective-writing is executed form a lower address to a higher address in one direction. Thereby, the numbers of writing times to all flash memory chips **40** in the flash memory **41** are averaged.

In the present embodiment, as described above, the numbers of writing times to the flash memory chips **40** in the flash memory **41** are averaged.

(Embodiment 9)

In the present embodiment, as shown in the embodiment 6, new parities are not sequentially, collectively written to the parity flahs memory chips for storing parities, but new parities are written to a plurality of flash memory chips **40** for storing parities in parallel. The present embodiment is constituted by applying the method shown in the embodiment 5 to the device shown in the embodiment 6; therefore, a detailed explanation thereof will be omitted.

What is claimed is:

1. A method of rewriting error correcting codes in a disc array device which includes a plurality of disc devices which hold a plurality of error correcting data groups, each of which includes a plurality of data and at least one error correcting code generated therefrom, the method comprising the steps of:

in response to a data rewrite request provided by an upper level device connected to said disc array device, reading old data designated by the data rewrite request from said plurality of disc devices;

reading, from said plurality of disc devices, an old error correcting code which belongs to one group within said plurality of error correcting data groups, to which one group the old data belongs;

generating a new error correcting code for the one error correcting data group, after the old data has been rewritten by new data designated by the data rewrite request, from the read old data, said old error correcting code and the new data;

rewriting the old data held in said plurality of disc devices by said new data;

temporarily holding the generated new error correcting code in a random access memory provided in said disc array device;

repeating said step of reading old data to said step of temporarily holding the new error correcting code for each of a plurality of other data rewrite requests subsequently provided by said upper level device, thereby storing a group of new error correcting codes for a group of data rewrite requests in said memory; and

sequentially writing the group of new error correcting codes held in said memory into a group of storage locations within said plurality of disc devices, at which storage locations a group of old error correcting codes have been held, according to an order of access predetermined for storage locations within the disc devices;

wherein the order of access is predetermined so that a plurality of storage locations which belong to the same track within one of said disc devices are sequentially accessed according to an order of locations within the track.

2. A method of rewriting error correcting codes according to claim **1**, wherein the order of access is predetermined so that a plurality of storage locations belonging to the same cylinder of one of said disc devices are accessed sequentially according to an order predetermined for tracks to which the plurality of storage locations belong, and so that a plurality of storage locations belonging to different cylinders of one of said disc devices are accessed sequentially according to an order predetermined for the cylinders.

3. A method of rewriting error correcting codes according to claim **1**, wherein the group of new error correcting codes held in said memory are sequentially written into the group of storage locations according to an order of generation of corresponding ones of the group of data rewrite requests issued by said upper level device.

4. A method of rewriting error correcting codes according to claim **1**, wherein the plurality of disc devices are divided into a plurality of data holding disc devices and at least one error correcting code holding disc device; and

wherein the reading of the group of old error correcting codes and the writing of the group of new error correcting codes are carried out in said error correcting code holding disc device.

5. A method of rewriting error correcting codes according to claim **4**, wherein the order of access is predetermined so that a plurality of storage locations belonging to the same cylinder of one of said disc devices are accessed sequentially according to an order predetermined for tracks to which the plurality of storage locations belong, and so that a plurality of storage locations belonging to different cylinders of one of said disc devices are accessed sequentially according to an order predetermined for the cylinders.

6. A method of rewriting error correcting codes according to claim **4**, wherein the group of new error correcting codes held in said memory are sequentially written into the group of storage locations according to an order of generation of corresponding ones of the group of data rewrite requests issued by said upper level device.

7. A method of rewriting error correcting codes according to claim **4**,

wherein said error correcting code holding disc device includes a plurality of areas;

wherein the group of new error correcting codes are held in said memory as a plurality of new error correcting code partial groups, each partial group corresponding to one of said areas and each partial group including plural new error correcting codes which correspond to plural old error correcting codes held at plural storage locations, each of which belong to the same one of said areas;

wherein the step of sequentially writing the group of new error correcting codes is executed, so that new error correcting codes belonging to different error correcting code partial groups are sequentially written, and so that new error correcting codes belonging to each partial group are written sequentially, according to said order of access, into a group of storage locations which belong to one of said areas corresponding to each partial group, and at which storage locations a plurality of old error correcting codes corresponding to a plurality of new error correcting codes belonging to each partial group are held.

8. A method of rewriting error correcting codes according to claim **7**, wherein the order of access is predetermined so that a plurality of storage locations belonging to the same cylinder within one of said areas within said error correcting code holding disc device are accessed sequentially according to an order predetermined for tracks to which the plurality of storage locations belong, and so that a plurality of storage locations belonging to different cylinders within said one area are accessed sequentially according to an order predetermined for the cylinders.

9. A method of rewriting error correcting codes according to claim 7, wherein the step of sequentially writing the group of new error correcting codes is executed so that a plurality of new error correcting codes belonging to each partial group and held in said memory are sequentially written into a group of storage locations within one of said areas corresponding to each partial area, according to an order of generation of corresponding ones of a group of data rewrite requests issued by said upper level device.

10. A method of rewriting error correcting codes according to claim 1, wherein the plurality of disc devices include a plurality of error correcting code holding areas provided in different ones of said disc devices;

wherein the group of new error correcting codes are held in said memory as a plurality of new error correcting code partial groups, each partial group corresponding to one of said areas and each partial group including plural new error correcting codes which correspond to plural old error correcting codes held at plural storage locations, each of which belong to the same one of said areas;

wherein the step of sequentially writing the group of new error correcting codes is executed so that new error correcting codes belonging to different error correcting code partial groups are sequentially written, and so that new error correcting codes belonging to each partial group are written sequentially, according to said order of access, into a group of storage locations which belong to one of said areas corresponding to each partial group, and at which storage locations a plurality of old error correcting codes corresponding to a plurality of new error correcting codes belonging to each partial group are held.

11. A method of rewriting error correcting codes according to claim 10, wherein the order of access is predetermined so that a plurality of storage locations belonging to the same cylinder within one of said areas within one of said disc devices are accessed sequentially according to an order predetermined for tracks to which the plurality of storage locations belong, and so that a plurality of storage locations belonging to different cylinders within said one area are accessed sequentially according to an order predetermined for the cylinders.

12. A method of rewriting error correcting codes according to claim 7, wherein a plurality of new error correcting codes belonging to each partial group and held in said memory are sequentially written into a group of storage locations within one of said areas corresponding to each partial area, according to an order of generation of corresponding ones of a group of data rewrite requests issued by said upper level device.

13. A method of rewriting error correcting codes according to claim 1, further comprising:

reading, from said plurality of disc devices to said memory, error correcting codes other than a group of old error correcting codes corresponding to the group of new error correcting codes held in said memory, after the repeating step;

wherein the step of sequentially writing the group of new error correcting codes is executed so that the group of new error correcting codes and said read other error correcting codes are sequentially written into a group of storage locations of said plurality of disc devices according to the order of access.

14. A method of rewriting error correcting codes according to claim 13, wherein each of the storage locations is one which holds either one of the group of new error correcting codes or one of the other group of error correcting codes.

15. A method of rewriting error correcting codes according to claim 4, further comprising:

reading, from said error correcting code holding disc device to said memory, error correcting codes other than a group of old error correcting codes corresponding to the group of new error correcting codes held in said memory, after the repeating step;

wherein the step of sequentially writing the group of new error correcting codes is executed so that the group of new error correcting codes and said read other error correcting codes are sequentially written into a group of storage locations of said error correcting code holding disc devices according to the order of access.

16. A method of rewriting error correcting codes according to claim 15, wherein each of the storage locations is one which holds either one of the group of new error correcting codes or one of the other group of error correcting codes.

17. A method of rewriting error correcting codes according to claim 7, further comprising:

reading, from each of the areas of said error correcting code holding disc device to said memory, error correcting codes other than a group of old error correcting codes corresponding to the partial group of new error correcting codes held in said memory in correspondence to said each area, after the repeating step;

wherein the step of sequentially writing the group of new error correcting codes is executed so that the partial group of new error correcting codes and said read other error correcting codes held in said memory in correspondence to each area are sequentially written into storage locations within each area of said error correcting code holding disc device according to the order of access, wherein each of the storage locations within each area is one which holds either one of the partial group of new error correcting codes for each area or one of the other group of error correcting codes for each area.

18. A method of rewriting error correcting codes according to claim 10, further comprising:

reading, from each of the error correcting code holding areas within said plurality of disc devices to said memory, error correcting codes other than a group of old error correcting codes corresponding to the group of new error correcting codes held in said memory in correspondence to each error correcting code holding area, after the repeating step;

wherein the step of sequentially writing the group of new error correcting codes is executed so that the group of new error correcting codes for each error correcting code holding area and said read other error correcting codes held in said memory in correspondence to each error correcting code holding area are sequentially written into storage locations within each error correcting code holding area, according to the order of access, wherein each of the storage locations within each area is one which holds either one of the group of new error correcting codes for each error correcting code holding area or one of the other group of error correcting codes for each error correcting code holding area.

19. A method of recovering error correcting codes according to claim 4, further comprising the steps of:

reading a plurality of error correcting codes from said error correcting code holding disc, at the occurrence of a fault during accessing of one of said plurality of data holding disc devices;

sequentially reading a plurality of groups of data from others of said data holding discs, other than said one faulty data holding disc, each group of data comprising data belonging to the same error correcting data group and being held at mutually the same address within said other data holding disc devices, said groups being read sequentially group by group;

selecting, from said plurality of error correcting codes as read, an error correcting code which should belong to the same error correcting data group as one to which each group within said groups of data as read belongs; and

recovering, for each group of data, data held in said faulty disc device and belonging to the same error correcting data group, from said error correcting code selected for each group and said read group of data.

20. A method of recovering error correcting codes according to claim **4**,

wherein said disc array device further includes;

a plurality of other disc devices including a plurality of other data holding disc devices and at least one other error correcting code holding disc device for said other data holding devices; and

a common error correcting code holding disc device provided for said plurality of disc devices and said plurality of other disc devices;

wherein the method further comprises the steps of:

in response to another group of data rewrite requests provided by said upper level device, executing the step of reading old data to said step of sequentially writing a plurality of old data and a plurality of new data, both related to said another group of data rewrite requests and held in said plurality of other disc devices, thereby generating another group of new error correcting codes for said plurality of other disc devices, and sequentially writing the generated another group of new error correcting codes into said other error correcting code holding disc device;

generating a still other group of new error correcting codes from the group of new error correcting codes generated for said plurality of disc devices and from said another group of new error correcting codes generated for said plurality of other disc devices;

writing the generated still other group of error correcting codes into said common error correcting code holding disc device;

sequentially reading a group of error correcting codes from one of said error correcting code holding disc device and said other error correcting code holding disc device, at the occurrence of fault with another one of said error correcting code holding disc device and said other error correcting code holding disc device;

sequentially reading said still other group of error correcting codes from said common error correcting code holding disc device; and

recovering a group of error correcting codes held in said faulty error correcting code holding disc device, from said group of error correcting codes read from said one error correcting code holding device and said still other group of error correcting codes read from said common error correcting code holding device.

21. A method of rewriting error correcting codes in a disc array device which includes a plurality of data holding disc devices and holds a plurality of error correcting data groups, each of which includes a plurality of data and at least one error correcting code generated therefrom, the method comprising the steps of:

in response to a data rewrite request provided by an upper level device connected to said disc array device, reading old data designated by the data rewrite request from one of said plurality of disc devices;

reading an old error correcting code which belongs to one group within said plurality of error correcting data groups to which one group the old data belongs, from a flash memory provided in said disc array device;

generating a new error correcting code for the one error correcting data group, after the old data has been rewritten by new data designated by the data rewrite request, from the read old data, said old error correcting code and the new data;

rewriting the old data held in said one disc device by said new data;

temporarily holding the generated new error correcting code in a random access memory provided in said disc array device;

repeating said step of reading old data to said step of holding a new error correcting code for each of a plurality of other data rewrite requests subsequently provided by said upper level device, thereby storing a group of new error correcting codes for a group of data rewrite requests into said random access memory;

erasing a group of old error correcting codes which correspond to said group of new error correcting codes from a group of storage locations within said flash memory, after the repeating step; and

sequentially writing the group of new error correcting codes held in said random access memory into the group of storage locations within said flash memory according to an order of access predetermined for storage locations within said flash memory.

22. A method of rewriting error correcting codes according to claim **21**, wherein the group of new error correcting codes held in said random access memory are sequentially written into the group of storage locations according to an order of generation of corresponding ones of the group of data rewrite requests issued by said upper level device.

23. A method of rewriting error correcting codes according to claim **21**,

wherein said flash memory includes a plurality of areas;

wherein the group of new error correcting codes are held in said random access memory as a plurality of new error correcting code partial groups, each partial group corresponding to one of said areas and each partial group including plural new error correcting codes which correspond to plural old error correcting codes held at plural storage locations each of which belong to the same one of the areas;

wherein the step of sequentially writing the group of new error correcting codes is executed so that error correcting codes belonging to different error correcting code partial groups are written sequentially, and so that new error correcting codes belonging to each partial group are written sequentially, according to said order of access, into a group of storage locations which belong to one of said areas corresponding to each partial group and at which storage locations a plurality of old error correcting codes corresponding to a plurality of new error correcting codes belonging to each partial group are held.

24. A method of rewriting an error correcting codes according to claim **21**, further comprising:

reading, from said flash memory to said random access memory, error correcting codes other than a group of old error correcting codes corresponding to the group of new error correcting codes held in said random access memory, after the repeating step; and

erasing the group of old error correcting codes and said other error correcting codes from said flash memory, after the reading of the latter;

wherein the step of sequentially writing the group of new error correcting codes is executed so that the group of new error correcting codes and said read other error correcting codes are sequentially written into a group of storage locations of said flash memory according to the order of access.

25. A method of rewriting error correcting codes in a disc array device which includes a plurality of data holding disc devices and which holds a plurality of error correcting data groups, each of which includes a plurality of data and at least one error correcting code generated therefrom, the method comprising the steps of:

in response to a data rewrite request provided by an upper level device connected to said disc array device, reading old data designated by the data rewrite request from one of said plurality of devices;

reading an old error correcting code which belongs to one group within said plurality of error correcting data groups, to which one group the old data belongs, from a flash memory provided in said disc array device;

generating a new error correcting code for the one error correcting data group after the old data has been rewritten by new data designated by the data rewrite request, from the read old data, said old error correcting code and the new data;

rewriting the old data held in said one disc device by said new data;

temporarily holding the generated new error correcting code in a random access memory provided in said disc array device;

repeating said step of reading old data to said step of temporarily holding the generated new error correcting code for each of a plurality of other data rewrite requests subsequentially provided by said upper level device, thereby storing a group of new error correcting codes for a group of data rewrite requests into said random access memory; and

executing an erasing operation, after the repeating step, to a group of storage locations within said flash memory which have successive addresses and hold invalid information;

sequentially writing the group of new error correcting codes held in said random access memory into the group of storage locations according to a predetermined order of addresses; and

repeating the step of reading old data to the step of sequentially writing the group of new error correcting codes into said flash memory for different groups of data rewrite requests provided by said upper level device;

wherein one group within different groups of storage locations within said flash memory which have successive addresses and which hold invalid information is selected from said flash memory at the writing of a group of new error correcting codes generated for each group within the other groups of data rewrite requests during the last mentioned repetition.

26. A disc array device, comprising:

a plurality of disc devices which hold a plurality of error correcting data groups, each of which includes a plurality of data and at least one error correcting code generated therefrom;

a disc array controller connected to said plurality of disc devices and an upper level device, said disc array controller including a random access memory and a control device;

said control device including:

means responsive to a data rewrite request provided by said upper level device for reading old data designated by the data rewrite request from said plurality of disc devices;

means for reading, from said plurality of disc devices, an old error correcting code which belongs to one group within said plurality of error correcting data groups to which one group the old data belongs;

means for generating a new error correcting code for the one error correcting data group, after the old data has been rewritten by new data designated by the data rewrite request, from the read old data, said old error correcting code and the new data;

means for rewriting the old data held in said plurality of disc devices by said new data;

means for writing the generated new error correcting code into the random access memory;

means for repetitively operating said first mentioned reading means to said last mentioned writing means for each of a plurality of other data rewrite requests subsequently provided by said upper device, thereby storing a group of new error correcting codes for a group of data rewrite requests in said random access memory; and

means for sequentially writing the group of new error correcting codes held in said random access memory into a group of storage locations within said plurality of disc devices, at which storage locations a group of old error correcting codes have been held, according to an order of access predetermined for storage locations within said disc devices;

wherein the order of access is predetermined so that a plurality of storage locations which belong to the same track within one of said disc devices are sequentially accessed according to an order of locations within the track.

27. A disc array device, comprising:

a plurality of data holding disc devices which hold a plurality data belonging to a plurality of error correcting data groups each of which includes a plurality of data and at least one error correcting code generated therefrom;

a flash memory for holding a plurality of error correcting codes for the plurality of error correcting data groups; and

a disc array controller connected to said plurality of disc devices, said flash memory and an upper device, said disc array controller including a random access memory and a control device;

said control device including:

means responsive to a data rewrite request provided by said upper device for reading old data designated by the data rewrite request from said plurality of disc devices;

means for reading from said flash memory an old error correcting code which belongs to one group within said plurality of error correcting data groups to which one group the old data belongs;

means for generating a new error correcting code for the one error correcting data group after the old data has been rewritten by new data designated by the data rewrite request, from the read old data, said old error correcting code and the new data;

means for rewriting the old data held in said plurality of disc device by said new data;

means for writing the generated new error correcting code into the random access memory;

means for repetitively operating said first mentioned reading means to mentioned writing means for each said last of a plurality of other data rewrite requests subsequently provided by said upper level device, thereby storing a group of new error correcting codes

for a group of data rewrite requests in said random access memory;

means for erasing a group of storage locations within said flash memory which hold a group of old error correcting codes corresponding to the group of new error correcting codes held in said random access memory; and

means for sequentially writing the group of new error correcting codes held in said random access memory into the group of storage locations within said flash memory, according to an order of access predetermined for storage locations within said flash memory.

* * * * *